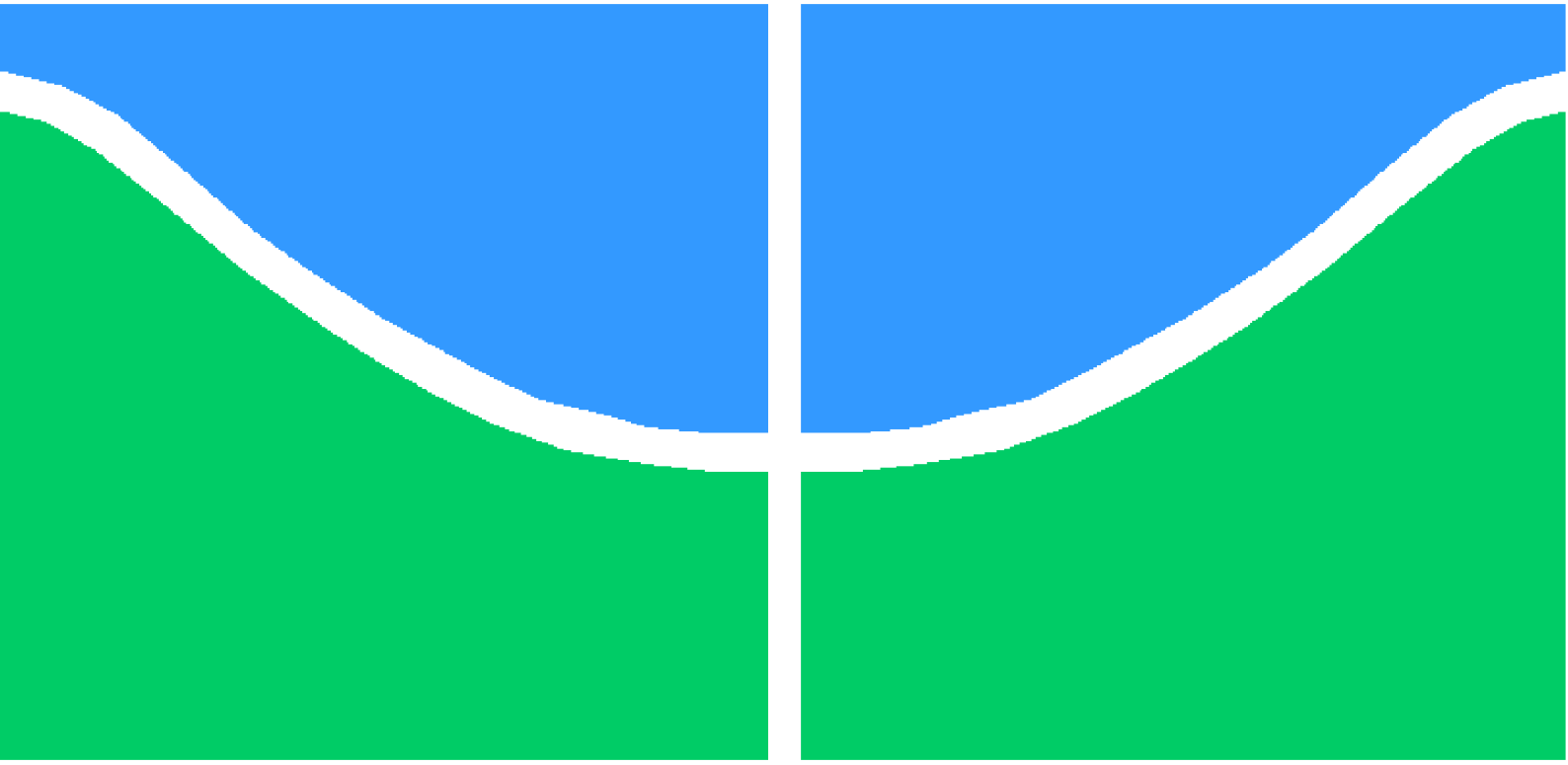

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Engenharia de Software

Comparação de Algoritmos para Qualificação das Saídas de Busca em Gerenciadores de Distribuição Linux

Autor: Luiz Fernando Gomes de Oliveira

Orientador: Dr. Edson Alves da Costa Júnior

Brasília, DF

2015

Luiz Fernando Gomes de Oliveira

Comparação de Algoritmos para Qualificação das Saídas de Busca em Gerenciadores de Distribuição Linux

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Dr. Edson Alves da Costa Júnior

Coorientador: Dr. Fábio Macedo Mendes

Brasília, DF

2015

Luiz Fernando Gomes de Oliveira

Comparação de Algoritmos para Qualificação das Saídas de Busca em Gerenciadores de Distribuição Linux/ Luiz Fernando Gomes de Oliveira. – Brasília, DF, 2015-

128 p. : il. (algumas color.) ; 30 cm.

Orientador: Dr. Edson Alves da Costa Júnior

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2015.

1. Gerenciadores de pacotes. 2. Contribuição em Código Aberto. I. Dr. Edson Alves da Costa Júnior. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Comparação de Algoritmos para Qualificação das Saídas de Busca em Gerenciadores de Distribuição Linux

CDU 02:141:005.6

Luiz Fernando Gomes de Oliveira

Comparação de Algoritmos para Qualificação das Saídas de Busca em Gerenciadores de Distribuição Linux

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Trabalho aprovado. Brasília, DF, 01 de junho de 2015:

Dr. Edson Alves da Costa Júnior
Orientador

Dr. Fábio Macedo Mendes
Convidado 1

Dr. Paulo Roberto Miranda Meirelles
Convidado 2

Brasília, DF
2015

Agradecimentos

Agradeço meus pais, irmã, padrinhos, madrinhas e demais membros da minha família, que me apoiaram nos meus momentos de glória e dificuldades. A minha namorada por ser compreensiva e abrir mão do nosso tempo juntos para a escrita deste trabalho assim como muitos outros já concluídos e sempre que pode, me ofereceu ajuda, conforto e incentivo. A todos os meus professores que souberam atuar não apenas como funcionários de uma universidade, mas como mestres e sobretudo, amigos, dentre eles, um agradecimento especial aos professores: Adson Rocha, Carla Rocha, Cristiano Miosso, Suelia Rodrigues e Paulo Meirelles, tal como não posso deixar de citar os amigos Bruno Santiago, João Cerqueira, Yuri Mota, Helbert Junior, Lucas Severo, Diogo Oliveira, Carlos Oliveira e Matheus Fernandes, que como irmãos, estiveram sempre ao meu lado e ofereceram a mão para ajudar sempre que precisei, assim como muitos outros que me vieram a cabeça, mas recebem minhas desculpas além dos agradecimento, ou esta lista ficaria mais extensa que todo o meu trabalho. Claro, não posso deixar de citar o jedi que me guiou por grande parte do curso ate aqui, obrigado Edson Júnior pelos esforços em educar mais um padawan na sua jornada.

"People like to invent monsters and monstrosities.
Then they seem less monstrous themselves."
Andrzej Sapkowski, The Last Wish

Resumo

Este trabalho vem apresentar a implementação de melhorias e acréscimo de opções para apresentação de resultados de buscas em gerenciadores de repositórios de pacotes presentes nas distribuições Linux, com o intuito de apresentar os resultados mais relevantes ao usuário com base nos valores de entrada, simplificar o processo de busca e tratar os casos onde a entrada possuir erros ortográficos, sejam por desconhecimento do nome do pacote, seja por casual escrita incorreta. Tal implementação foi construída a partir de algoritmos de *string matching* presentes na literatura para a ampliação do funcionamento do APT.

Palavras-chaves: Linux. Gerenciadores de pacotes. APT. C++. Python. Comunidade Código Aberto.

Abstract

This work is presenting the implementation of improvements and increased options for presenting search results in package repositories managers present in Linux distributions, in order to present the most relevant results to the user based on the input values, simplify the process search and address cases where entry has misspellings, whether through ignorance of the package name, either casual misspellings. Such implementation was built from string matching algorithms in the literature to expand the operation of the APT.

Key-words: Linux. Packages Managers. APT. C++. Python. Open Source Community.

Lista de ilustrações

Lista de tabelas

Lista de códigos

Lista de abreviaturas e siglas

APT	*Advanced Packaging Tool* - Ferramenta de Empacotamento Avançada.
CLI	*Command Line Interface* - Interface de Linha de Comando.
KMP	Knuth–Morris–Pratt - Algoritmo de autoria de Donald Knuth, Vaughan Pratt e James H. Morris.
PR	*Pull Request*, muitas vezes utiliza-se o termo *Merge Request* - Envio de trecho de código para uma contribuição.

Sumário

Introdução

Este documento apresenta considerações gerais e preliminares relacionadas ao uso de gerenciadores de pacotes e repositórios em sistemas Linux. São abordados exemplos de uso e suas possíveis dificuldades para novos usuários ao sistema aberto quando são posicionados à rotina do uso de um terminal para a inserção de comandos para a realização de tarefas.

Objetivos

Primeiramente, este trabalho tem como objetivo a avaliação de diferentes algoritmos de *string matching*, com o intuito de evidenciar o potencialde melhoria dos gerenciadores de pacotes atuais em relação à apresentação de resultados de pesquisa por pacotes. Consciente do objetivo deste trabalho, será evitado o desenvolvimento dos códigos de algoritmo de ordenação, visto que o objetivo não é a implementação dos algoritmos, mas a comparação dos resultados obtidos através dos algoritmos de *string matching* e suas performances nos estados atuais de implementação. Assim, serão utilizadas bibliotecas que possuem implementações dos referidos algoritmos na fase de prototipação, citando suas respectivas referências e créditos, e a implementação que ocorrerá na etapa final do trabalho será baseada na disponibilização dos algoritmos em bibliotecas públicas e suas adaptações para o contexto do trabalho.

Objetivo Geral

Este trabalho tem como objetivo apresentar métodos de qualificação dos resultados de busca de modo que apresentem resultados mais próximos do esperado pelos usuários, quando estes forem realizar a busca por pacotes disponíveis para instalação em suas distribuições, tendo como foco o APT como gerenciador principal.

Objetivos Específicos

- Implementar ou usar bibliotecas que implementam algoritmos de *string matching* que venham a qualificar a saída apresentada em uma busca por pacotes.

- Implementar protótipos de qualificação da saída das buscas que torne a localização dos pacotes desejados menos dependentes de ferramentas complementares como `grep`, `more` ou `less`.

- Implementar protótipos que possibilitem a apresentação de resultados de uma busca por pacotes, mesmo quando inserido o nome de um pacote inexistente ou com erros ortográficos.

- Realizar contribuições com o APT com o intuito de implementar melhorias no código para melhorar a qualidade da saída por busca de pacotes.

O uso de gerenciadores

Um dos processos que novos usuários Linux tem dificuldade em se adaptar é a instalação de novas aplicações. Hoje é comum haver nas plataformas uma aplicação que centraliza a instalação das demais aplicações. Visto como uma loja em algumas plataformas, o gerenciador de aplicações tão comum nos dias atuais era, há alguns anos um diferencial dos sistemas Linux. Usuários do sistema operacional Windows que iniciavam o aprendizado do sistema Linux tinham dificuldade para compreender que não precisavam acessar o site de um fabricante para fazer o *download* de aplicações diversas. Tudo o que precisavam era abrir o terminal e com alguns comandos o software era instalado de uma fonte confiável.

A compreensão deste processo abria novas fronteiras para o usuário, que agora buscava expandir seu conhecimento e testar novas aplicações dentre as milhares que estavam à sua disposição. Obviamente que o processo de instalação e remoção de pacotes eventualmente torna-se dependente de uma ferramenta que possibilite realizar buscas por aplicações e filtragem das aplicações retornadas. E este é um ponto onde os novos usuários encontravam dificuldades.

As primeiras dificuldades

Quando realizadas em linha de comando, as buscas por pacotes podem ser complexas para usuários inexperientes em relação ao uso de comandos no terminal, e a ordenação dos resultados é ainda menos intuitiva. Uma busca por *"pdf"* no `apt` leva hoje a um resultado que, além de estourar o *buffer* de linhas da janela do terminal, apresenta o popular `evince` aproximadamente como 14ª opção dentre os pacotes disponíveis para instalação, como por ser observado na Figura 1, dependendo das listas de repositórios adicionados na distribuição e qual gerenciador esta sendo usado.

Alem disso, aplicações que não tem relação com leitores de *pdf*, tais como o antivírus `clamav` ou o produtor de documentação `doxygen`, aparecem antes do `evince` devido às regras de ordenação do APT. O primeiro parâmetro de ordenação do `APT` é o repositório em que o pacote esta localizado. Essa ordem é definida de acordo com os dados inseridos no `/etc/apt/sources.list` e `/etc/apt/sources.list.d`, sendo esta etapa sensível à ordem

```
luiz@DESKTOP:~/Documents/UnB/TCC/TCC1$ apt-cache search pdf | more
apparmor-docs - Documentation for AppArmor
bison-doc - Documentation for the Bison parser generator
clamav - anti-virus utility for Unix - command-line interface
cups-filters - OpenPrinting CUPS Filters - Main Package
cups-filters-core-drivers - OpenPrinting CUPS Filters - PPD-less printing
dblatex - Produces DVI, PostScript, PDF documents from DocBook sources
debiandoc-sgml - DebianDoc SGML DTD and formatting tools
docbook-utils - Convert DocBook files to other formats (HTML, RTF, PS, man, PDF)
doxygen - Documentation system for C, C++, Java, Python and other languages
doxygen-dbg - Debug symbols for doxygen
doxygen-doc - Documentation for doxygen
erlang-doc - Erlang/OTP HTML/PDF documentation
erlang-erl-docgen - Erlang/OTP documentation stylesheets
evince - Document (PostScript, PDF) viewer
evince-common - Document (PostScript, PDF) viewer - common files
evince-dbg - Document (PostScript, PDF) viewer - debugging symbols
fop - XML formatter driven by XSL Formatting Objects (XSL-FO.)
fop-doc - XML formatter driven by XSL Formatting Objects (doc)
ghostscript - interpreter for the PostScript language and for PDF
ghostscript-dbg - interpreter for the PostScript language and for PDF - Debug symbols
ghostscript-doc - interpreter for the PostScript language and for PDF - Documentation
ghostscript-x - interpreter for the PostScript language and for PDF - X11 support
```

Figura 1 – Apresentação de resultados da busca do termo *pdf* usando o `apt-cache`

da listagem dos repositórios nos arquivos. O segundo parâmetro de ordenação do pacotes é a ordenação alfabética dos pacotes selecionados em um mesmo repositório serão ordenados alfabeticamente. Os pacotes são selecionados por terem, seja em seu nome, descrição ou *tags*, a sequência *"pdf"*.

```
luiz@DESKTOP:~/Documents/UnB/TCC/TCC1$ apt search pdf | more

WARNING: apt does not have a stable CLI interface yet. Use with caution in scripts.

Sorting...
Full Text Search...
ada-reference-manual-2005/trusty 1:2012.2-2ubuntu1 all
  Ada 2005 language standard

ada-reference-manual-2012/trusty 1:2012.2-2ubuntu1 all
  Ada 2012 language standard

aephea/trusty 12-248-2 all
  text-based authoring tool for HTML

aliki/trusty 0.3.0-1 amd64
  Measurement tool for Impulse Responses

aliki-dbg/trusty 0.3.0-1 amd64
  Debugging symbols for aliki

altree-examples/trusty 1.3.1-1 all
  example files for ALTree

antiword/trusty,now 0.37-9 amd64 [installed,automatic]
  Converts MS Word files to text, PS and PDF

apparmor-docs/trusty 2.8.95~2430-0ubuntu5 all
  Documentation for AppArmor

apsfilter/trusty 7.2.6-1.3 all
  Magic print filter with automatic file type recognition

apt-dpkg-ref/trusty 5.3.1ubuntu1 all
  APT, Dpkg Quick Reference sheet

apvlv/trusty 0.1.1-1.2 amd64
  PDF viewer with Vim-like behaviour

asis-doc/trusty 2010-6 all
  Ada Semantic Interface Specification (ASIS) documentation

auto-multiple-choice-doc-pdf/trusty 1.2.1-1 all
```

Figura 2 – Apresentação de resultados da busca do termo *pdf* usando o `apt`

O resultado é ainda menos preciso quando utilizada a interface do `apt` ao invés do `apt-cache`. Como esta interface apresenta todos os resultados em uma única ordenação

alfabética, temos na 11ª posição um visualizador de PDF (*apvlv*), porém o popular `evince` nem mesmo entra na lista dos 20 primeiros.

Dentre os diversos gerenciadores de repositório utilizados atualmente para a instalação de pacotes em distribuições Linux, podemos apontar alguns que se destacam por virem instalados de padrão nas distribuições mais comuns hoje e, consequentemente sendo os mais utilizados, como o `apt` ou o `yum`.

Porém, os principais gerenciadores possuem em comum o mesmo problema de ordenação dos resultados de uma busca. A apresentação de resultados de uma busca não é personalizável, não permitindo classificar a listagem de pacotes por nome, por exemplo, nem fazem *matching* de aproximação entre o pacote pesquisado e os possíveis candidatos. Desta forma a procura por um pacote do qual não se tenha o nome correto pode se tornar cansativa e desgastante, especialmente para um novo usuário Linux, que desconhece de ferramentas como o `grep, more` ou `less`.

Parte I

Desenvolvimento

1 Fundamentação Teórica

Neste capitulo será apresentado os conceitos e referências utilizadas para a construção deste trabalho. Alguns algoritmos apontados neste trabalho foram publicados com objetivos distintos dos utilizados atualmente ou eram unicamente teóricos devido ao contexto da época onde a computação e os avanços tecnológicos eram motivos de limitação para uma aplicação real destes algoritmos. Dos que foram designados para objetivos distintos de aproximação de *strings*, é notável quantos deles surgiram da área de biologia, no intuito de semelhanças de especies ou proximidades genéticas.

Para as contribuições para código aberto, a filosofia apresentada pela comunidade é a de trabalho em conjunto, com produção em massa de diversos pontos de vista distintos, a fim de oferecer a melhor funcionalidade e manter um alto grau de confiabilidade em segurança.

1.1 Gerenciadores de pacotes

Desde o lançamento das distribuições *Debian* e *Slackware* em 1993, diversas distribuições tiveram seu ciclo de vida realizado por completo, apresentando e amadurecendo a filosofia por trás da distribuição, gerando novas ramificações e por fim sendo abandonadas. Hoje, se escolhermos alguma distribuição Linux dentre as inúmeras existentes, provavelmente ela será descendente de uma das principais distribuições: *Debian, Red Hat* ou *Slackware*. As Figuras 14 e 15, no Anexo A, apresenta um modelo de herança entre as distribuições Linux onde pode-se observar a influência que estas distribuições tiveram em relação a um grupo de novas distribuições que se ramificaram delas[1].

Em sistemas operacionais Linux há muito foi proposto uma alternativa para distribuições de aplicações para o sistemas: os pacotes. A filosofia por trás da comunidade livre (BRETTHAUER, 2001) e o interesse em buscar novas formas de armazenamento e distribuição de pacotes fez com que surgissem diversos *gerenciadores de pacotes*, voltados à determinadas distribuição cada (BECK; MAGNUS; KUNITZ, 2002).

Devido grande parte das distribuições hoje existentes serem ramificações das distribuições *Debian, Red Hat* ou *Slackware*, os diversos gerenciadores de pacotes hoje disponíveis também possuem muitas similaridades entre eles. É o caso do *dpkg* e o *rpm*, onde as funcionalidades de instalação, remoção e provimento de informações de pacotes estão presentes em ambos. Ambos também são escritos predominantemente em *C* e *Perl*.

[1] Apesar da difícil visualização das imagens no documento, o intuito das figuras apresentadas é apresentar a vasta quantidade de distribuições que surgiram de bifurcações das distribuições *Debian* e *Red Hat*, tal como alterações nelas provavelmente viriam a influenciar as demais distribuições.

Suas diferenças surgem quando chegamos a averiguar quais pacotes cada aplicação suporta. O *dpkg* suporta os pacotes *.deb*, enquanto o *rpm* está voltado para os pacotes *rpm*. Como há uma distinção em como estes pacotes são montados (BAILEY, 1997), há uma incompatibilidade entre as duas aplicações.

Gerenciadores de Pacotes e Gerenciadores de Repositórios

Um ponto de vista importante de ser caracterizado neste trabalho é a diferença entre os termos *gerenciadores de pacotes* e *gerenciadores de repositórios*. É importante caracterizar a diferença estes dois termos comumente utilizados como sinônimos devido a sua similaridade. A proximidade é tamanha que na própria documentação oficial do *Debian* não há uma caracterização de *gerenciadores de repositórios*, citando apenas que gerenciadores como o `aptitude`[2] ou `dselect`[3] dependem do `APT`, que por sua vez depende do `dpkg`[4] (DEBIAN, 2013). Essa cadeia de dependências indicam as camadas de interface, onde o `dpkg` é responsável pela instalação, remoção e gerência dos pacotes, enquanto o `APT` está relacionado aos repositórios onde são armazenados os pacotes. Já aplicações como o `aptitude` ou o `synaptic` visam oferecer uma camada *user friendly*, com interfaces gráficas e menos informações textuais.

Assim, definimos o termo *gerenciadores de pacotes* àquelas aplicações dedicadas ao manuseio dos pacotes (instalação e remoção, por exemplo), enquanto *gerenciadores de repositórios* são as aplicações voltadas ao controle da lista de repositórios de pacotes e interface para instalação, remoção e pesquisa, fazendo de uso dos *gerenciadores de pacotes* quando necessário.

Tendo em vista que este trabalho busca apresentar uma melhor apresentação dos resultados de uma busca por um pacote a ser instalado em uma pesquisa realizada em um gerenciador de repositórios, devemos inicialmente elencar os possíveis candidatos a serem avaliados no trabalho.

Comandos básicos

Por serem baseados em uma lista de repositórios, os *gerenciadores de repositórios* recomendam uma atualização de seus bancos de dados antes de realizar qualquer sequência de operações para garantir que seus *links* serão válidos. A Tabela 3 apresenta os comandos para a atualização ou sincronização da lista de repositórios e pacotes dos gerenciadores.

[2] Gerenciador de repositórios CLI com suporte a interface amigável ainda em terminal. Mais informações podem ser obtidas no manual oficial, disponível em <https://www.debian.org/doc/manuals/aptitude/>

[3] Gerenciador de repositórios CLI mais simples e atualmente suplantado pelo `APT`, caindo em desuso. Mais informações podem ser obtidas no manual oficial, disponível em <https://www.debian.org/doc/manuals/dselect-beginner/>

[4] Gerenciador de pacotes CLI. Mais informações podem ser obtidas na documentação oficial, disponível em <http://man.cx/dpkg(8)>

Os comandos de atualização de lista de repositórios precisam de permissão administrativa e devem ser relizados como usuário *root* ou ter a invocação do comando `sudo` antes do comando de instalação/remoção. Seguindo a padronização adotada pelas distribuições Linux para apresentação de comandos em terminal (HEKMAN; ORAM, 1996), comandos precedidos de `#` implicam em comandos que necessitam de permissões administrativas enquanto comandos precedidos por `$` podem ser efetuados por qualquer usuário.

Tabela 1 – Comandos para atualização do banco de dados dos gerenciadores de repositórios mais populares.

Gerenciador de Repositórios	Commando
APT	`# apt-get update`
YUM	`# yum update`
Emerge	`# emerge -sync`

Pressupondo que os repositórios já estejam sincronizados na maquina local, é possível realizar os comandos básicos de instalação, remoção ou busca por pacotes sem riscos de que um pacote não seja encontrado ou tenha sua referência incompleta ou inválida. Estes comandos devem ser passados também via terminal. Os comandos de instalação e remoção de pacotes precisam de permissão administrativa. A Tabela 2 lista estes comandos para os gerenciadores de repositórios mais populares, onde `<pacote>` é o nome do pacote com o qual se deseja operar. É importante frisar que tem-se por padrão a nomeação dos pacotes sem caracteres especiais ou espaços. Apesar de existir uma padronização quanto ao versionamento dos pacotes e suas arquiteturas, não é necessário a caracterização destas informações. Para operações de instalação de pacotes, serão priorizados os pacotes mais recentes com arquitetura compatível com a máquina, sendo necessário a descriminação da arquitetura `x86` para a instalação de pacotes 32 *bits* em sistemas 64 *bits*.

Tabela 2 – Principais comandos dos gerenciadores de repositórios mais populares.

Gerenciador	Instalar	Remover	Procurar
APT	`# apt-get install <pacote>`	`# apt-get purge <pacote>`	`$ apt-cache search <pacote>`
YUM	`# yum install <pacote>`	`# yum purge <pacote>`	`$ yum search <pacote>`
Emerge	`# emerge <pacote>`	`# emerge remove <pacote>`	`$ emerge search <pacote>`

1.2 APT

O `APT`, *Advanced Package Tool, Ferramenta de Empacotamento Avançada*, é um gerenciador de repositórios amplamente utilizado em distribuições oriundas da Debian (*Debian-like*). É provavelmente o gerenciador mais comum atualmente devido a popularidade de distribuições como **Ubuntu**, **Mint** e **Debian**. Projetado inicialmente para

substituir o gerenciador `dselect`[5], o `APT` teve suas primeiras *builds* distribuídas via IRC em Agosto de 1998, sendo integrado ao *Debian* na *release* de Março de 1999 (GARBEE et al., 2008). O `APT` pode ser considerado como uma interface para o `dpkg`, gerenciando as dependências de um pacote para a instalação e remoção, alem de apresentar uma relação dos pacotes disponíveis na lista de repositórios. Uma das possibilidades que o `APT` possui hoje é a de se realizar buscas utilizando de expressões regulares.

Um dos recursos que o `APT` fornece de forma transparente para o usuário é a ordenação de pacotes para instalação ou remoção com o uso das chamadas do `dpkg`, suprindo as dependências dos pacotes que são requisitos para o pacote ao qual o usuário deseja instalar na máquina, de forma que não venha a instalar um pacote de dependência quebrada. Dentre as desvantagens que o `APT` pode apresentar, as que se destacam são justamente o fato de estar escrito em C++, que apesar do ganho de performance que apresenta em relação aos seus concorrentes, carrega junto uma maior dificuldade de manutenção devido ao tamanho e complexidade.

Outro problema do `APT` é sua fragmentação. Apesar de serem tratadas como parte do APT, as aplicações como `apt-get` ou `apt-cache` são na realidade aplicações que fazem de uso do `APT` como uma biblioteca para interface, mas são comumente tratadas como aplicações integradas ao APT. Como consequências, o `APT` possuiu uma das interfaces menos intuitivas quando comparado com gerenciadores como o *YUM* ou o *Portage* justamente por frequentemente serem apresentadas soluções utilizando as aplicações como o `apt-get` ao invés do `apt`.

Por ser um *gerenciador de repositórios*, o `APT` trabalha unicamente com os arquivos *.deb* que possuam seus respectivos endereços registrados no arquivo `/etc/apt/sources.list`, possibilitando o uso de diretórios remotos ou locais para a disponibilização dos pacotes a serem gerenciados. Requisitada a instalação de um pacote ao `APT`, será feito o *download* do pacote e suas respectivas dependências e estes arquivos são repassados ao `dpkg` para que seja realizado o processo de instalação.

O `APT` possui como repositório oficial o <git://anonscm.debian.org/apt/apt.git>, apesar de também ser hospedado em outros repositórios, tal como o <https://github.com/Debian/apt>, de forma espelhada. Neste trabalho estaremos desenvolvendo soluções que venham a contribuir e enriquecer a aplicação `apt` , em especial o comando **search**. Segundo podemos observar o código disponibilizado do *APT*, a apresentação dos resultados de uma busca são ordenados alfabeticamente, de acordo com a lista de repositórios disponíveis.

[5] O `dselect` é uma interface em console para o gerenciamento de pacotes Debian. Altamente utilizada na decada de 90, atualmente é uma ferramenta que um dos manuais do Debian (BORTZMEYER, 1999) desestimula o uso.

Interfaces

Atualmente existem varias interfaces para o uso do APT. Apesar de aplicações como *Synaptic, Aptitude* e *Adept Package Manager* serem bastantes usuais e conhecidas, há diversos *bindings* disponíveis para integrar o `APT` junto a outras ferramentas, sendo o python-apt[6] um dos mais comuns devido sua estabilidade e simplicidade. Outra interface interessante é o apt-rpm[7], uma interface para o uso de pacotes distribuidos em RPM, para distribuições como Fedora, Red Hat, SuSE, ALT-Linux, etc (APT-RPM, 2010).

Comandos

Assim como diversas outras aplicações CLI, *Command Line Interface, Interface de Linha de Comando*, o `APT` possui comandos que podem ser passados como parâmetros em linha de execução. Segue eles:

Tabela 3 – Parâmetros disponíveis para uso da interface CLI do APT.

Parâmetro	**Ação**
`autoclean`	Apaga arquivos antigos já baixados.
`autoremove`	Remove automaticamente todos os pacotes não utilizados.
`build`	Constrói pacotes localmente a partir dos códigos fontes.
`build-dep`	Configura dependências para pacotes.
`changelog`	Apresenta histórico de modificações de um pacote.
`check`	Verifica se há alguma dependência quebrada.
`clean`	Apaga arquivos temporários baixados.
`contains`	Lista pacotes contidos em um arquivo.
`content`	Lista arquivos contidos em um pacote.
`deb`	Instala um arquivo *.deb*.
`depends`	Apresenta dependências de um pacote.
`dist-upgrade`	Realiza uma atualização de versão do sistema operacional, com instalações e remoções necessárias.
`download`	Baixa o arquivo *.deb* de um pacote.
`dselect-upgrade`	Segue as seleções do comando `dselect`.
`held`	Lista todos os pacotes guardados.
`help`	Apresenta a ajuda de um comando específico.
`hold`	Guarda um pacote.
`install`	Instala ou atualiza pacotes, com suas dependências.
`policy`	Apresenta politicas de configurações.
`purge`	Remove pacotes e seus arquivos de configuração.

[6] Disponível em <https://apt.alioth.debian.org/python-apt-doc/library/index.html>
[7] Disponível em <http://apt-rpm.org/about.shtml>

`rdepends`	Apresenta dependências reversas de um pacote.
`reinstall`	Reinstala um pacote já instalado.
`remove`	Remove um pacote.
`search`	Busca por pacotes utilizando nomes ou expressões.
`show`	Mostra informações detalhadas de um pacote.
`source`	Baixa código fonte de um pacote.
`sources`	Edita o arquivo `/etc/apt/sources.list` com editor padrão.
`unhold`	Remove um pacote guardado.
`update`	Atualiza cache com lista de pacotes disponíveis.
`upgrade`	Realiza a atualização de um pacote.
`version`	Apresenta a versão instalada de um pacote.

1.3 Algoritmos de *string matching*

Nessa seção será apresentados algoritmos de *string matching* presentes na literatura que foram utilizados na fase de prototipação ou durante a etapa de implementação e contribuição. Eles estão divididos em dois grupos principais. O primeiro grupo é para a localização de uma sequencia de palavras em um texto e estarão classificados como *string matching* exatos neste trabalho. O segundo grupo é formado por algoritmos que apontam a diferença entre duas palavras. Eles são utilizados para realizar buscas de palavras em um texto quando se é considerado um possível erro ortográfico e estarão classificados como *string matching* inexatos neste trabalho.

1.3.1 Algoritmos de *string matching* exatos

Há inúmeros exemplos possíveis de algoritmos voltados para *string matching* exatos, mas iremos citar e apresentar dois dos mais importantes.

Rabin-Karp

Criado em 1987 por Richard M. Karp e Michael O. Rabin, o algoritmo faz de uso de uma *hash* com o intuito de agilizar a busca por um padrão em um texto. Para busca em um texto de tamanho n por um padrão de tamanho p, o algoritmo de Rabin-Karp pode ter em melhor caso ordem de complexidade $\Theta(n-m+1)$ (FEOFILOFF, 2015)[8]. Em pior caso, a ordem de complexidade chega a $\Theta((n-m+1)m)$, mesma ordem de complexidade que uma busca comum/força bruta (FEOFILOFF, 2015). Todavia, a situação de pior caso

[8] Onde $m = n + p$

só ocorre quando a *hash* não é propriamente calculada. No Código 1.1 apresentamos de forma simples como implementar o algoritmo de Rabin–Karp.

Código 1.1 – Algoritmo de Rabin–Karp

```
1  def string_matching_rabin_karp(text='', pattern='', hash_base=256):
2      n = len(text)
3      m = len(pattern)
4      offsets = []
5      htext = hash_value(text[:m], hash_base)
6      hpattern = hash_value(pattern, hash_base)
7      for i in range(n-m+1):
8          if htext == hpattern:
9              if text[i:i+m] == pattern:
10                 offsets.append(i)
11         if i < n-m:
12             htext = (hash_base *
13                      (htext -
14                       (ord(text[i]) *
15                        (hash_base ** (m-1))))) + ord(text[i+m])
16     return offsets
```

Observe que na linha 5 do Código 1.1 temos o cálculo de uma *hash* para o texto, *hash* esta que será aproveitada para o calculo de uma nova *hash* na linha 15 do código. Este é o conceito de *rolling hash*, possibilitando obter um novo valor da *hash* com seu valor antigo, nos poupando tempo de iterações e evitando que a linha 5 seja executada em casos de chances nulas. Este é o ponto onde o algoritmo de Rabin–Karp se mostra com alto desempenho. Normalmente, o algoritmo é utilizado com uma *rolling hash* simples que faz de uso de multiplicações e somas apenas, conforme apresentada na Equação 1.1,

$$H = c_1 a^{k-1} + c_2 a^{k-2} + c_3 a^{k-3} + ... + c_k a^0 \tag{1.1}$$

onde a é uma constante e $c_1, ...c_k$ são os caracteres de entrada. Para evitar o cálculo de números muito grandes, normalmente é usado aritimética em $\mod m$ para o valor dos índices. A chave para uma boa *hash* está na escolha dos valores de m e a que satisfaçam a arquitetura utilizada para os cálculos e que se enquadrem nas condições recomendadas para uma congruência linear (KNUTH, 1998), o que nessa *hash* implica que $0 < a < m$.

Knuth–Morris–Pratt (KMP)

O algoritmo publicado em 1977 tem autoria de Donald Knuth, Vaughan Pratt e James H. Morris e faz de uso do conceito de autômato de estados, ou seja, uma máquina abstrata que deve estar em um, e apenas um, de seus finitos estados (THIERBACH, 1985). Este algoritmo segue do pressuposto que o texto é um fluxo contínuo (FEOFILOFF,

2015), evitando de voltar a pontos anteriores do texto após uma falha de verificação, mas sim executar os seguintes passos quando encontrarmos um conflito entre `texto[i]` e `pattern[j]`:

1. Localizar o maior k tal que `pattern[0..k-1]` é igual a `texto[i-k+1..i]`.
2. Seguir a comparação de `texto[i+1..]` com `pattern[k..]`.

Por considerar sempre um fluxo continuo do texto, a ordem de complexidade do algoritmo de Knuth–Morris–Pratt é previsível: $\Theta(n)$, porém requer um pré-processamento com ordem $\Theta(m)$ (HUME; SUNDAY, 1991), tornando o algoritmo com complexidade $\Theta(mn)$. No Código 1.2 podemos observar o algoritmo de Knuth–Morris–Pratt implementado em linguagem Python.

Código 1.2 – Knuth–Morris–Pratt (KMP)

```
1  def kmp(pattern):
2      if len(pattern) == 0:
3          return None
4      T = [0] * len(pattern)
5      T[0] = -1
6      pos = 2
7      cnd = 0
8      while pos < len(pattern):
9          if pattern[pos - 1] == pattern[cnd]:
10             cnd += 1
11             T[pos] = cnd
12             pos += 1
13         elif cnd > 0:
14             cnd = T[cnd]
15         else:
16             T[pos] = 0
17             pos += 1
18     return T
```

1.3.2 Algoritmos de *string matching* inexatos

Leveinstein

Duas das abordagens propostas neste trabalho são baseadas nos trabalhos de *Vladimir Levenshtein*. Atuante na área de teoria da informação, código de correção de erros e análise combinatória, Levenshtein publicou em 1965 na *Problemy Peredachi Informatsii* o trabalho pelo qual seria reconhecido desde então: *Binary codes with correction of drooping and insertion of the symbol 1* (LEVENSHTEIN, 1965). Republicado num resumo em

1966 com o título de *Binary codes capable of correcting deletions, insertions and reversals* (LEVENSHTEIN, 1966), a obra tratava do cálculo da distância entre duas *strings*, ou seja, dada as *strings* **A** e **B**, determinar a quantidade mínima de operações necessárias para que, partindo da *string* **A**, seja possível obter a *string* **B**. Entende-se por operações a inserção, remoção e substituição de um caractere.

Exemplo de cálculo de distância entre *strings*

Para uma maior compreensão do que se trata uma operação e a distância de Levenshtein entre *strings*, tomemos como exemplos duas palavras, **cantar** e **dançar**. Vamos aplicar algumas operações para, a partir da primeira, alcançar a segunda palavra.

0. cantar
1. dantar *(Substituição de "c" por "d")*
2. dançar *(Substituição de "t" por "ç")*

Como podemos observar, foram necessárias duas operações para alcançar a palavra **dançar**. Como este é o número mínimo de operações para se obter o resultado desejado, a distância entre elas é 2. É importante observar observar que, mesmo que duas letras iguais e consecutivas sejam substituídas por outras duas letras idênticas, serão creditadas duas operações, por exemplo as palavras **correr** e **Potter**:

0. correr
1. Porrer *(Substituição de "c" por "P")*
2. Potrer *(Substituição do primeiro "r" por "t")*
3. Potter *(Substituição do segundo "r" por "t")*

Precisamos então de 3 operações para alcançar **Potter** partindo de **correr**. É do pressuposto que o algoritmo é *case-sensitive* e sua ordenação é baseada na Tabela *ASCII (American Standard Code for Information Interchange)*, tabela inicialmente baseada no alfabeto inglês com codificação de *7-bits* por caractere em um intervalo de até 128 caracteres, imprimíveis ou não (SHIREY, 2007).

Distância de Levenshtein

Nos trabalhos de Levenshtein de 1965 e 1966, são apesentadas apenas as possibilidades de localização de um caractere perdido, ou do *bit* perdido na sequência binária.

Porém a descrição apresentada para a identificação do caractere no trabalho de 1966 (LEVENSHTEIN, 1966) possibilitou a implementação do algoritmo no decorrer da história, tendo uma das sua representações mostrada no Código 1.4.

Código 1.3 – Implementação da distância de Levenshtein

```
def calculate_distance(s1,s2):
    d = {}
    lenstr1 = len(s1)
    lenstr2 = len(s2)
    for i in range(-1,lenstr1):
        d[(i,-1)] = i+1
    for j in range(-1,lenstr2):
        d[(-1,j)] = j+1

    for i in range(lenstr1):
        for j in range(lenstr2):
            if s1[i] == s2[j]:
                cost = 0
            else:
                cost = 1
            d[(i,j)] = min(
                           d[(i-1,j)] + 1, # deletion
                           d[(i,j-1)] + 1, # insertion
                           d[(i-1,j-1)] + cost, # substitution
                          )

    return d[lenstr1,lenstr2]
```

Como pode-se observar, a implementação do cálculo da distância utiliza programação dinâmica a partir de operações de inserção, remoção e substituição. Tal abordagem facilita a inserção de novas operações, conforme apresentado na próxima subseção.

Damerau-Levenshtein

A distância de Damerau-Levenshtein é uma variação da distância de Levenshtein (LEVENSHTEIN, 1965), havendo ainda a adição da transposição no conjunto de operações básicas. Em seus estudos, Damerau apresentou que mais de 80% dos erros de escrita oriundo de humanos estavam relacionados às operações de falta de caractere, excesso de caracteres, substituição de caracteres ou transposição de caracteres (DAMERAU, 1964).

Por considerar a transposição de dois caracteres adjacentes, a distância de *Damerau-Levenshtein* abriu portas para um novo padrão de métricas, as biológicas, no campo de mensurar a variação entre moléculas de DNA. Esta seria a brecha necessária para que surgissem outros métodos de análise de moléculas de DNA baseadas na comparação de

strings, tais como o algoritmo de *Needleman–Wunsch* (NEEDLEMAN; WUNSCH, 1970) e o de *Smith-Waterman* (SMITH; WATERMAN, 1981).

Implementação

Para melhor contextualização da diferença entre o método tradicional de cálculo de distância e o modelo de *Damerau-Levenshtein*, consideremos a seguir uma visão atômica de como seria o procedimento do cálculo da distância de Levenshtein, conforme mostrado no Código 1.4.

Código 1.4 – Visão atômica do cálculo da distância de Levenshtein

```
1 def levenshtein_distance(s1, s2):
2     lenstr1 = len(s1)
3     lenstr2 = len(s2)
4     d = initialize_distance(lenstr1,lenstr2)
5
6     d = calculate_distance(lenstr1,lenstr2,d)
7
8     return d[lenstr1-1,lenstr2-1]
```

O Código 1.4 serviria tanto para o modelo tradicional como para o de *Damerau-Levenshtein*, necessitando apenas implementar o método `calculate_distance()` de forma distinta. O Código 1.5 representa como seria a implementação para o modelo de *Damerau-Levenshtein*.

Código 1.5 – Implementação da distância de Damerau-Levenshtein

```
1 def calculate_distance(lenstr1,lenstr2,d):
2     for i in xrange(lenstr1):
3         for j in xrange(lenstr2):
4             if s1[i] == s2[j]:
5                 cost = 0
6             else:
7                 cost = 1
8             d[(i,j)] = min(
9                         d[(i-1,j)] + 1, # deletion
10                        d[(i,j-1)] + 1, # insertion
11                        d[(i-1,j-1)] + cost, # substitution
12                       )
13            # transposition
14            if i and j and s1[i]==s2[j-1] and s1[i-1] == s2[j]:
15                d[(i,j)] = min (d[(i,j)], d[i-2,j-2] + cost)
16
17    return d
```

A distinção do método de *Damerau-Levenshtein* para o modelo tradicional é apenas a inserção das linhas **13** e **14** do Código 1.5, que realiza a transposição dos caracteres, quando for o caso, ao custo de uma única operação.

Coeficiente de Sørensen–Dice

Também conhecido como *índice de Sørensen* ou *coeficiente de Dice*, o algoritmo que faz de uso de análise estatística foi independentemente desenvolvido por Dice (1945) e Sørensen (1948) para análise de estudos amostras de especies. Porém, semelhante ao acontecido com o algoritmo de Smith-Waterman (SMITH; WATERMAN, 1981), o modelo apontado por *Lee Dice* e *Thorvald Sørensen* acabou sendo abstraído para conjuntos de caracteres e fatalmente utilizado para o apontamento de semelhanças entre *strings*. O modelo propõe o uso de *n-grams* (CAVNAR; TRENKLE et al., 1994) para calcular a proximidade entre duas *strings* contando as intersecções entre elas. Tradicionalmente são utilizados bigramas, grupos de dois caracteres, para a execução do algoritmo, porém a restrição do tamanho dos grupo implica apenas na flexibilidade com que a semelhança entre os grupos serão analisados. Com o valor das intersecções entre as *strings*, o índice de coincidência é calculado de acordo com a Equação 1.2, onde $\mid X \mid$ e $\mid Y \mid$ representam o tamanho dos *n-grams* de cada *string* e o módulo dos conjuntos representa o tamanho e $\mid X \cap Y \mid$ o tamanho da intersecção, desconsiderando repetições.

$$s = \frac{2 \mid X \cap Y \mid}{\mid X \mid + \mid Y \mid} \tag{1.2}$$

Por haver uma comparação de *n-grams* nas duas *strings*, a ordem com que elas aparecem não influenciam no resultado final, porém há a possibilidade deste algoritmo pode vir a ter ordem de complexidade $\Theta(n)$ dependendo da sua implementação, tendo assim um desempenho semelhante ao das expressões regulares. A Figura 3 apresenta um comparativo entre as *strings* `ALGORITHMS ARE FUN` e `LOGARITHMS ARE NOT`. Observe que o *bigram* `AR` é selecionado em ambos.

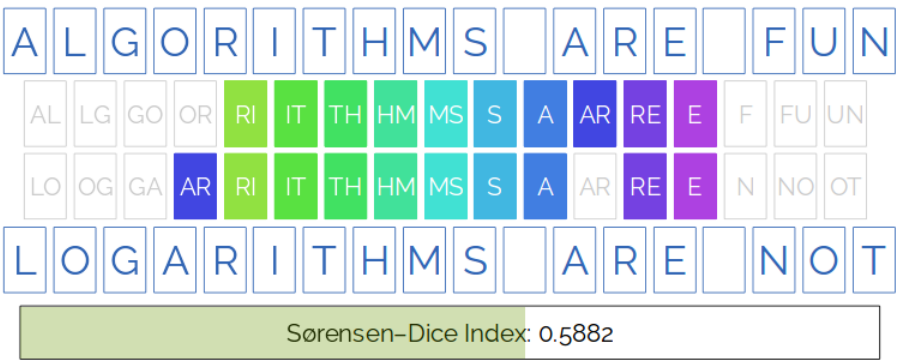

Figura 3 – Representação da saída do algoritmo de Sørensen–Dice[9].

[9] **Fonte:** http://www.algomation.com/algorithm/sorensen-dice-string-similarity.

O Código 1.6 apresenta uma solução simples e não otimizada para o calculo do coeficiente de Sørensen–Dice na linguagem Python com o uso de `sets` para o cálculo do coeficiente em unigramas. Apesar de não ser uma solução ortodoxa do algoritmo e permitir falhas como apontar que as *strings* `AA` e `AAAA` como idênticas, a solução é interessante para apresentar de forma direta o funcionamento do algoritmo, visto a simplicidade que ele passa a ter quando implementado com `sets` e utilizando do operador lógico & para identificar as interseções entre as *strings*.

Código 1.6 – Simples implementação do coeficiente de Sørensen–Dice

```
1 def dice_coefficient(a, b):
2     """dice coefficient 2nt/na + nb."""
3     a_unigram = set(a)
4     b_unigram = set(b)
5     overlap = len(a_unigram & b_unigram)
6     return overlap * 2.0/(len(a_unigram) + len(b_unigram))
```

Smith-Waterman

Proposto em 1981 por *Temple F. Smith* e *Michael S. Waterman* (SMITH; WATERMAN, 1981) com o intuito de localizar sequências moleculares semelhantes, este algoritmo de programação dinâmica é hoje amplamente utilizado para a localização de regiões similares entre *strings* e nas sequências de proteínas ou nucleotídeos.

O algoritmo trabalha com o alinhamento de *strings* no intuito de buscar a maior semelhança entre elas. Considerando este fator, é interessante que as duas *strings* contenham uma quantidade de caracteres igual ou próxima, visto que a *string* com maior quantidade de caracteres provavelmente terá suas bordas ignoradas no final. Como resultado, podemos obter melhores resultados com comparações de *strings* que os algoritmos de *Leveinstein* quando tratamos de busca por radicais. Todavia uma comparação usando palavras com uma diferença significativa na quantidade de caracteres poderá resultar em uma perda de desempenho devido ao número de permutações necessárias para localizar o melhor alinhamento.

Algoritmo

De acordo com o artigo de Smith e Waterman (SMITH; WATERMAN, 1981), o algoritmo pode ser apresentado da seguinte forma: Considerando as *strings* a e b de tamanhos n, m respectivamente e $s(a, b)$ a função de similaridade do universo a ser con-

siderado[10], é montada uma matriz H para encontrar os dois pares com maior grau de similaridade sendo W_i a nota de lacuna[11].

Valores preliminares de H tem a interpretação de que $H(i,j)$ tem a máxima similaridade de duas *strings* que possuem final a_i e b_j respectivamente. Os valores de $H(i,j)$ podem ser obtidos através das Equações 1.3 e 1.3:

$$\begin{aligned} H(i,0) &= 0, \quad 0 \leq i \leq m \\ H(0,j) &= 0, \quad 0 \leq j \leq n \\ H(i,j) &= \max\{A(i,j), B(i,j), C(i,j), 0\}, \quad 1 \leq i \leq n \text{ e } 1 \leq j \leq m \end{aligned}$$

onde

$$\begin{aligned} A(i,j) &= H(i-1, j-1) + s(a_i, b_j) \\ B(i,j) &= \max_{k \geq 1} \{H(i-k, j) - W_k\} \\ C(i,j) &= \max_{l \geq 1} \{H(i, j-l) - W_l\} \end{aligned}$$

Montada a matriz H, para obter o melhor alinhamento, o algoritmo começa com o maior valor na matriz com índice (i,j), retornando em sentido a célula $(0,0)$, seguindo pelos pontos $(i-1,j), (i,j-1)$, ou $(i-1,j-1)$.

A fórmula para $H(i,j)$ considera as possibilidades de fim da *string* para qualquer a_i ou b_j, tal que:

1. Se a_i e b_j forem associados, a similaridade será dada pela Equação 1.3:

$$H_{i-1,j-1} + s(a_i, b_j) \tag{1.3}$$

2. Se a_i está ao fim de uma remoção de largura k, a similaridade será dada pela Equação 1.4:

$$H_{i-k,j} - W_k \tag{1.4}$$

3. Se b_i está ao fim de uma remoção de largura l, a similaridade será dada pela Equação 1.5:

$$H_{i,j-l} - W_l \tag{1.5}$$

4. Por fim são incluídos sentinelas a fim de evitar o cálculo de similaridades negativas.

[10] Smith considerava apenas as letras *A,C,U,G* devido ao algoritmo ser voltado para identificação de similidades genéticas, todavia a generalização para as demais letras do alfabeto não invalida o algoritmo.

[11] Também conhecida como *gap-scoring* ou *gap-penalty*.

2 Metodologia e Procedimentos

Neste capitulo será apresentada a forma como os experimentos serão realizados, assim como o ambiente utilizado para obter estes resultados. Por se tratar de uma contribuição dividida em etapas cíclicas, elas serão apresentadas divididas em sessões.

2.1 Metodologia de trabalho

Um dos objetivos deste trabalho era contribuir com uma aplicação *open-souce* disponível na plataforma GitHub[1]. Na próxima seção será apresentada a forma como tais contribuições foram realizadas. Devido ao fato do repositório do `APT` estar armazenado no GitHub, a terminologia apresentada estará em maior conformidade com esta plataforma, porém o conceito de *pull-request* ou *merge-request* é amplamente utilizado em outras plataformas, como Bitbucket[2], GitLab[3] ou até mesmo SourceForge[4].

Antes das contribuições, analises com algoritmos de *string matching* foram realizadas com o uso da linguagem Python com o intuito de prototipação de uma solução para a busca de pacotes com o APT. Nesta prototipação, as funções foram implementadas com o uso de bibliotecas que já tinham a implementação dos algoritmos, visto que o intuito do trabalho não é a implementação dos algoritmos, mas a avaliação do desempenho alcançado com o uso deles.

Algoritmos Propostos para Prototipação em Python

Sendo o interesse do estudo apresentar alternativas para a qualificação dos resultados de uma busca apresentado hoje no gerenciador e não novos algoritmos de aproximação de *string*, foram priorizados algoritmos de implementação pouco rebuscada, que apresentassem bons resultados de comparação de *strings* e que já estivessem implementados em linguagem *Python*. Desta forma, foram escolhidos três algoritmos disponibilizados no PyPI. São eles:

swalign Pacote que oferece o algoritmo de comparação de *Smith-Waterman*. Utilizada a versão 0.3.1.

python-Levenshtein Pacote que oferece o algoritmo de comparação de *Levenshtein*. Utilizada a versão 0.11.2.

[1] Disponível em <http://github.com>.
[2] Disponível em <https://bitbucket.org>.
[3] Disponível em <http://gitlab.com>.
[4] Disponível em <http://sourceforge.net>.

pyxDamerauLevenshtein Pacote que oferece o algoritmo de comparação de *Damerau-Levenshtein*. Utilizada a versão 1.2.

Também foi utilizado o algoritmo de *match* exato, no qual deve-se encontrar a *string* procurada na íntegra no nome do pacote. Esta busca já é implementada no processo de *search* do APT, porém ela aparece em um segundo patamar de ordenação. Assim, foi implementado também um método de comparação baseada apenas no *match* exato da *string* de entrada em relação ao nome do pacote.

Definido os algoritmos que seriam utilizados no estudo, iniciou-se então a fase de prototipagem de código. O primeiro passo foi a familiarização com o pacote `apt`, disponibilizado por padrão na instalação do `python2.7` nas distribuições *Debian-like*. Este pacote possibilita a busca, instalação e remoção de pacotes. Para tal foi utilizada a classe `apt.cache.Cache` que nos possibilita verificar a existência de um pacote na lista de candidatos por meio da interface `__getitem__` padrão do *Python*. O Código A.1 apresenta a versão final deste protótipo, onde se era testado por padrão no método a procura e instalação do pacote `git` enquanto no método principal se testava as mesmas rotinas para o inexistente pacote `gito`.

O estudo deste pacote possibilitou a evolução de um esqueleto de código que permitiria o teste dos distintos algoritmos de *string-matching*. O primeiro a ser escrito foi o de *match* exato, apresentado no Código A.2. Este esqueleto oferecia elementos básicos dos protótipos que seriam criados a seguir. São eles:

Classe Pack Classe básica construída para armazenar os resultados obtidos dos algoritmos e posteriormente ordená-los. É formada basicamente pelo método de formatação de impressão do pacote, o nome do pacote, o *ratio* do pacote (usado para classificação) e os métodos que podem vir a ser utilizados pelas rotinas de ordenação de listas.

_parser Objeto responsável por realizar o *parser* das aplicações, oferecendo *flags* de controle, como quantidade máxima a ser listada no resultado, *ratio* mínimo a ser impresso, inclusão de prefixos e sufixos comuns ao nome do pacote procurado para classificação e a opção de executar um *pool* de *threads* na rotina a fim de reduzir o tempo de execução. Esta última opção foi classificada como depreciada devido a instabilidade que gerava no momento da ordenação dos pacotes.

Este modelo foi fundamental para a escrita dos outros três protótipos deste trabalho, apresentados nos Código A.3, Código A.4 e Código A.5 do Apêndice A deste documento. Como a estrutura do protótipo era mantida para os demais algoritmos, o único momento de retrabalho foi com o algoritmo de *Smith-Waterman*, onde o pacote

retornava uma lista de valores que deviam ser classificados, o que necessitou a reescrita de um método de ordenação dos pacotes, já que para o algoritmo oferecido pelo pacote em especial não havia um retorno de uma simples distância ou *ratio* de aproximação.

Os comandos foram todos executados em primeiro plano sem outra aplicação rodando em conjunto e foram-se efetuados cinco medidas consecutivas do tempo e retirada a mediana entre elas. A medida de tempo foi obtida executando a chamada das aplicações precedidas do comando `time` e tomado como resultado o valor retornado para o usuário. Na Figura 4 é possível observar um exemplo da procura pelo pacote *xpto* no `apt`. Neste exemplo, o valor considerado foi o de 890 ms.

```
luiz@DESKTOP:~/Documents/UnB/TCC/TCC1$ time apt search xpto
Sorting... Done
Full Text Search... Done

real    0m0.924s
user    0m0.890s
sys     0m0.033s
```

Figura 4 – Exemplo do uso do comando `time`

Mantendo uma cópia para a contribuição

Normalmente não temos autorização pra alterar as informações de outra pessoa sem seu o consentimento. Da mesma forma acontece com os repositórios `git`. Assim, um dos primeiros passos para começar a contribuir com uma ferramenta *open-source* é fazer uma cópia dela em seu repositório. Essa cópia pode ser feita basicamente de duas formas, no intuito de preservar os autores das modificações anteriores:

Fork : O termo é amplamente utilizado quando queremos fazer uma copia de um repositório para a nossa conta mantendo a rastreabilidade do projeto original. Normalmente este passo é realizado via interface gráfica, ainda no navegador. Na Figura 5 é possível observar o botão no canto superior direito que permite realizar um *fork* do repositório desejado.

Adicionando remote : Quando desejamos realizar uma cópia do repositório em uma plataforma distinta da original, a forma mais simples de proceder é clonando o repositório e incluindo nele um novo *remote* para a nova plataforma. O *remote* é o *link* do repositório utilizado para a transmissão das alterações. Ele pode oferecer uma conexão `hhtps` ou `ssh`. Assim conseguimos trabalhar com a mesma árvore de *commits* do repositório original, mantendo os créditos e alterações originais.

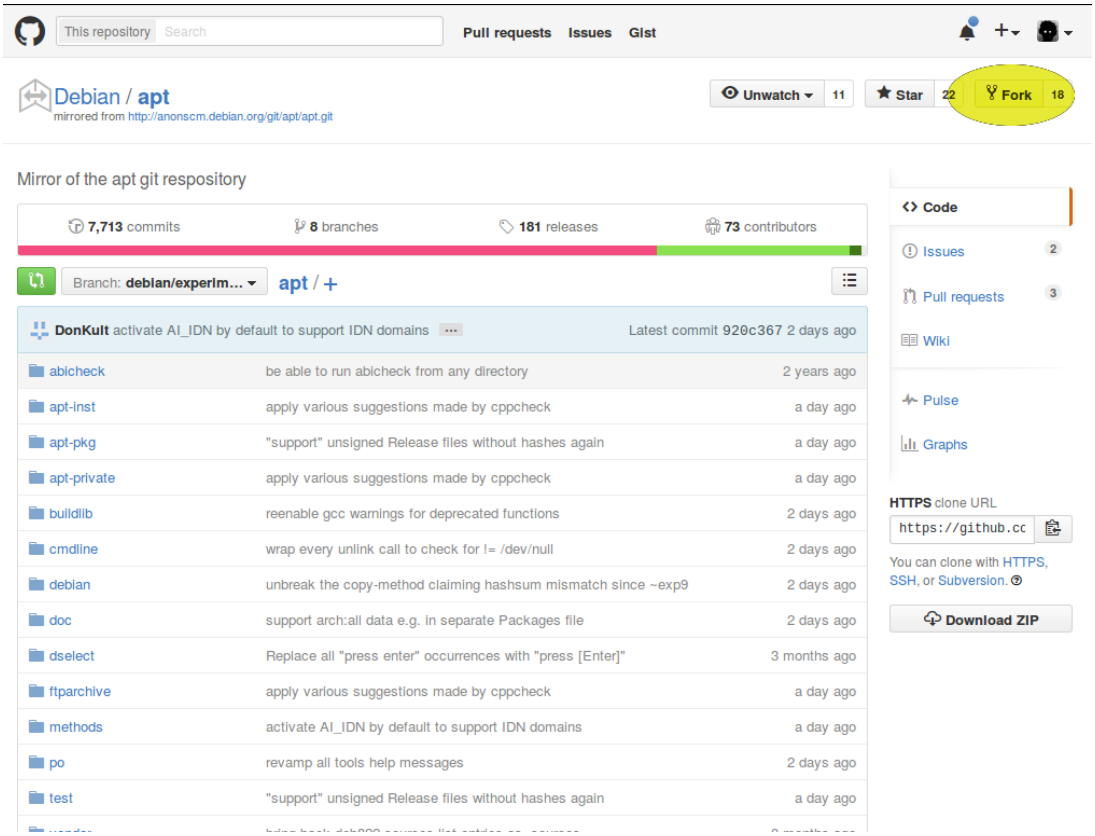

Figura 5 – Visão de um repositório do *GitHub*, com destaque no botão que permite criar um *fork* do projeto

Evoluindo o código

Nesto ponto de contribuição, o contribuinte desenvolve o código de acordo com o processo de produção escolhido. O código estará versionado em um repositório pessoal e o usuário possuirá todos os direitos administrativos deste repositório. Obviamente, caso a contribuição esteja sendo feita à uma aplicação grande e com muitos contribuidores, é importante se manter sincronizado com o código original, para evitar que o retorno da contribuição não seja tão trabalhoso para o revisor, além de evitar situações de conflito.

Retornando sua contribuição

Quando se decidido que a contribuição atinge um ponto aceitável para ser incorporada ao código original, é o momento de se realizar o *pull-request*[5]. Uma contribuição é interessante quando possui as seguintes características:

- É uma significativa e relevante evolução de código mantendo os padrões de guia de estilo definido pelo desenvolvedor original/linguagem;
- Adiciona ou atende os testes para a evolução de código proposto;

[5] Ou *merge-request* dependendo da plataforma de desenvolvimento.

- Traz a documentação da *feature* implementada ou atualiza a documentação para contemplar as modificações sugeridas.

Submetida a contribuição, é dever do mantenedor do repositório original revisar o código e decidir se deve ou não aceitá-la.

Testando

É desejável, no desenvolvimento de software, existir um conjunto de testes unitários a cada evolução de código, a fim de validar as funcionalidades existentes ou adicionadas. No projeto abordado neste trabalho, em especial, há testes utilizado o Google C++ Testing Framework[6] (Google Test), um *framework* desenvolvido pela Google para testes. Ele tem suporte para um conjunto de assertivas pré-definidas e definidas pelo usuário, tornando-se uma das melhores ferramentas para a escrita de conjuntos de testes para C/C++ hoje disponíveis.

Outra categoria de testes se refere aos testes de integração. Neste trabalho em especial estará sendo utilizado um *shell script* que oferece diversas funcionalidades para a criação dos testes de integração. Os testes de integração são utilizados no `APT` como testes caixa preta, a fim de verificar as funcionalidades em cenários de testes.

Os testes de integração tem maior prioridade que os unitários no APT, e podemos observar isso pela quantidade de testes em cada um deles. Nos testes unitários temos 20 casos de teste, totalizando 78 testes. Já nos testes de integração temos um conjunto de 199 cenários de testes, em um total de 24.976 testes[7].

Código 2.1 – Teste de validação de armazenamento de parâmetros

```
1  TEST(CommandLineTest,SaveInConfig)
2  {
3     EXPECT_CMD("apt-get install -sf",
4       "apt-get", "install", "-sf");
5     EXPECT_CMD("apt-cache -s apt -so Debug::test=Test",
6       "apt-cache", "-s", "apt", "-so", "Debug::test=Test");
7     EXPECT_CMD("apt-cache -s apt -so Debug::test=\"Das ist ein Test\"",
8       "apt-cache", "-s", "apt", "-so", "Debug::test=Das ist ein Test");
9     EXPECT_CMD("apt-cache -s apt --hallo test=1.0",
10      "apt-cache", "-s", "apt", "--hallo", "test=1.0");
11 }
```

Um exemplo de teste unitário realizado é a validação do armazenamento dos parâmetros nas chamadas, como pode ser observado no Código 2.1. Já testes de integração

[6] Disponível em <https://github.com/google/googletest>.
[7] Dados relativos a *build* #219 do Travis-CI, em <https://travis-ci.org/Debian/apt/builds/>.

possuem um comparativo entre uma chamada e sua saída, como pode ser observado no Código 2.2.

Código 2.2 – Teste de verificação de saída para busca

```
1 # without op progress
2 testsuccessequal "foo/unstable 1.0 all
3   $DESCR
4 " apt search -qq xxyyzz
```

2.1.1 Coleta de Dados

Tempo

A coleta de tempo de performance de um software é uma tarefa árdua. A estimativa de tempo depende das otimizações que o compilador pode vir a fazer para a arquitetura, memória disponível, aplicativos rodando em *background*, temperatura do hardware, etc. No intuito de simplificar o processo de estimativa de tempo, foi utilizado a mediana de uma série de execuções da aplicação por ser um bom parâmetro para custo linear. Assim, o Código B.1 foi uma solução desenvolvida para deixar a maquina dedicada para a aquisição de dados com base no nome das *branchs* e na lista de pacotes selecionados.

Para o funcionamento do *script*, as variáveis `MAX` e `_MAX_THREADS` devem ser definidas com a quantidade de dados que se deseja coletar e a quantidade máxima de *threads* que devem ser criadas para a chamada do processo, respectivamente. Usar uma quantidade superior à quantidade de *cores* da máquina pode produzir valores com baixa confiabilidade, visto a concorrência gerada pelas próprias *threads*. Como parâmetros, foram escolhidos 150 dados com 7 *threads*, deixando ao menos 1 *core* livre. Para maior dinamismo na coleta dos dados, o *script* é responsável por realizar a troca das *branchs* onde estão localizadas as possíveis soluções e a compilação e registro dos dados em uma planilha do *LibreOffice Calc*. Assim, os algoritmos nunca são executados paralelamente, porém os dados serão coletado sob a demanda da *CPU* na qual o *script* é executado.

Para coletar o tempo de execução do algoritmo, o Código B.2 foi desenvolvido a fim de ser usado como cronômetro. O algoritmo desenvolvido com o auxilio da biblioteca `Chrono`[8] permite a captura do intervalo de tempo com até 9 casas decimais (nanosegundos), todavia foi utilizado a precisão de microssegundos (10^6) apenas, visto que o intervalo de nanosegundos não implicou em diferença significativa dos valores. Os métodos `begin()` e `end()` da classe são utilizados para pontuar os intervalos onde o tempo deve ser registrado. O resultado da medição pode ser obtido com o retorno do método `end()` ou com a chamada do `currenttime()`, dedicado apenas ao retorno deste valor.

[8] Disponibilizada no `C++11` sob o *namespace* `std::crono`.

Memória

Para a coleta de memória, foi utilizado o `Valgrind`, com o auxílio da ferramenta `Massif`. Devido a criação da *hash* ser feita de forma algébrica e o método de *KMP* utilizar um autômato finito determinístico, a repetição prévia das buscas com apenas 10 amostras confirmavam a suspeita de não haver a variação do gasto de memória.

Para a execução da aplicação com o suporte ao `Valgrind` para análise do uso de memória, foi utilizado o comando:

```
$ valgrind --tool=massif ./apt search pacote [--regex]
```

2.1.2 Ferramentas

Para a realização desta contribuição as seguintes tecnologias estiveram envolvidas:

Sublime-Text Editor de texto.

Vim Editor de texto.

Git Sistema de controle de versão.

Make Ferramenta de execução de comandos `bash`.

GCC Compilador de C/C++.

Mint Distribuição Linux baseada em Debian.

Os serviços das seguintes organizações também foram utilizados na produção deste trabalho:

Github Oferecem serviço de hospedagem de repositórios.

Travis CI Oferecem serviço de Integração Continua.

Para desenvolvimento, foi utilizado uma maquina com as seguintes configurações:

OS: Mint 17.1 Rebecca

Kernel: x86_64 Linux 3.13.0-37-generic

CPU: Intel Core i7 CPU 870 @ 2.934GHz

GPU: GeForce GTX 465

RAM: 12Gb

Durante o desenvolvimento, as seguintes versões das ferramentas foram utilizadas:

GCC - 4.8.4

GIT - 1.9.1

Make - 3.81

APT

- **local:** 1.0.1ubuntu2
- **repositório:** 1.1 exp14

2.2 Procedimentos

Como planejamento geral após a fase de prototipação em Python, foram definidos três grandes conjuntos de contribuições com o `APT`. A primeira contribuição tinha como objetivos a familiarização com o ambiente e com o time oficial de desenvolvimento. Para isso ela deveria conter uma contribuição simples, porém que viesse a oferecer maiores oportunidades posteriormente. A segunda contribuição tinha como objetivo a implantação dos algoritmos de *string matching* exatos estudados sem descartar o modelo original de uso de expressões regulares para busca, visto que este modelo esta implementado utilizando funções nativas de $C++$ e assim tendo ótima peformace visto o custo do algoritmo. Para a terceira e ultima contribuição o planejamento era a implantação do algoritmo de buscas inexatas em *strings* para ser executado quando a pesquisa não retornasse nenhum resultado.

2.2.1 Prototipação em Python

Para validação dos protótipos, foram escolhidos 7 (sete) dentre os 30 (trinta) pacotes mais instalados segundo o Popcorn-Ubuntu[9] e realizado o teste de busca com cada um destes e os protótipos. Com base nos tempos, que estão apresentados na Seção 3.3 na Tabela 4, o algoritmo de Levenshtein foi selecionado para a próxima fase do trabalho, onde três algoritmos de busca inexatas seriam implementados e comparados durante as etapas de contribuição com o APT.

2.2.2 Primeira contribuição

Como tarefa, foi definido a implementação mínima de uma estrutura que permitisse um contexto de ordenação que pudesse ser definido pelo usuário. O intuito era oferecer as seguintes opções de ordenação:

[9] Disponível em <http://popcon.ubuntu.com/>.

Alphabetic Ordenação padrão em ordem alfabética. Os pacotes são ordenados de acordo com seu nome.

Reverse Alphabetic Semelhante a ordenação alfabética, porém em ordem decrescente, ou seja, palavras que começam com `Z` são apresentadas antes das iniciadas com `A`.

Status Ordena os pacotes de acordo com as seguintes características, na ordem apresentada:

1. desinstalado,
2. instalado e com possível atualização,
3. instalado via pacote local,
4. instalado e descartável (auto removível),
5. instalado automaticamente,
6. instalado,
7. atualizável,
8. com configuração residual

Version Ordena a saída de acordo com a versão do pacote.

Alteração da estrutura de dados

Originalmente, os dados eram armazenados em um `std::map` padrão do `C++`, visto que essa estrutura não permite a duplicação de chaves e possui uma ordenação alfabética subjacente. Todavia, a alteração do algoritmo de ordenação de um `std::map` deve ser feita em sua declaração, o que inviabiliza a escolha da ordenação através de uma *flag* passada como parâmetro de execução. Visto essa condição, a estrutura foi alterada para `std::vector`, por ser mais flexível quanto a sua forma de ordenação.

Naturalmente, a inserção de candidatos à saída do `APT` ocorre em dois níveis. O primeiro nível ordena de acordo com os repositórios inseridos no `/etc/apt/sources.list` da distribuição. O segundo nível ordena cada bloco de pacotes recebidos de um repositório em ordem alfabética. Isso implica que apenas adicionar candidatos ao vetor resultaria em uma ordenação com baixa precisão, visto que a sequência de repositórios pode variar de usuário para usuário. Assim, a troca de estrutura de dados de `std::map` para `std::vector` implica na necessidade de implementar o método de ordenação alfabética, nativamente implementado na estrutura `std::map`.

Um método para ordenação alfabética permite também ordenação alfabética reversa, visto que basta utilizar as referências do vetor inversas, ao invés da normais. Assim, foi necessário a escrita de três métodos, no total, para a primeira contribuição. O Código 2.3 apresenta como a seleção de ordenação é realizada.

Código 2.3 – Tomada de decisão de ordenação

```
1  switch(PackageInfo::getOrderByOption())
2  {
3     case PackageInfo::REVERSEALPHABETIC:
4        std::sort(outputVector.rbegin(), outputVector.rend(), OrderByAlphabetic)
             ;
5        break;
6     case PackageInfo::STATUS:
7           std::sort(outputVector.begin(), outputVector.end(), OrderByStatus);
8           break;
9     case PackageInfo::VERSION:
10       std::sort(outputVector.begin(), outputVector.end(), OrderByVersion);
11       break;
12    default:
13       std::sort(outputVector.begin(), outputVector.end(), OrderByAlphabetic);
14       break;
15 }
```

A versão apresentada neste trabalho é o resultado de alguns *feedback* sugeridos pelos mantenedores do projeto, em especial David Kalnischkies, o qual fez questão de pontuar detalhes na contribuição que poderiam ser melhorados ou questionados no processo de aceitação da contribuição.

Testando alterações

Realizar uma contribuição que não ofereça testes ou atenda aos testes significa oferecer um trabalho sem garantias de funcionalidade. Desta forma, as contribuições realizadas neste trabalho também estiveram sempre acompanhadas de seus respectivos testes. O `APT` utiliza uma ferramenta de testes muito comum em $C++$, a *Google Test*[10]. Uma *suíte* de testes é caracterizada pelo *Google Test* como um arquivo contendo N testes. Para manter o contexto, as condições da *suíte* de testes devem ser definidas no início do arquivo.

Código 2.4 – Declarações de instâncias para o teste

```
1 setupenvironment
2 configarchitecture "i386"
3
4 DESCR='Some description that has a unusual word xxyyzz and aabbcc and a UPPERCASE
      '
```

[10] Mais informações sobre a ferramenta podem ser obtidas em seu repositório oficial: <https://github.com/google/googletest>.

```
DESCR2='Some other description with the unusual aabbcc only'
DESCR3='Some package description'
DESCR4='an autogenerated dummy baz=0.1/installed'
insertpackage 'unstable' 'foo' 'all' '1.0' '' '' "$DESCR
 Long description of stuff and such, with lines
 .
 and paragraphs and everything."
insertpackage 'testing' 'bar' 'i386' '2.0' '' '' "$DESCR2"
insertpackage 'stable' 'package' 'all' '1.5' '' '' "$DESCR3"
insertinstalledpackage 'baz' 'all' '0.1'

setupaptarchive
```

Como apresentado no Código 2.4, algumas macros foram criadas pela equipe do `APT` para simplificar o processo de declaração de instâncias. Com estas instâncias declaradas, o contexto do teste é estabelecido e está pronto para rodar a *suite* de testes. Os testes são basicamente cenários do estado do sistema, como pode ser observado no Código 2.5.

Código 2.5 – Teste de busca pelo pacote *foo*

```
# search name
testsuccessequal "foo/unstable 1.0 all
  $DESCR
" apt search -qq foo
```

Como estamos ordenando pacotes, era importante que se pudesse realizar uma busca por uma grande quantidade de pacotes. Utilizando o recurso de busca com expressões regulares do APT foi possível montar uma lista maior de pacotes e realizar uma busca por um termo genérico, a fim de obter como resultado toda a lista de pacotes. O exemplo o Código 2.6 demostra este resultado, onde a busca é feita utilizando a *regex* `\w`, a fim de selecionar todos os pacotes que possuam pelo menos um caractere em seu nome.

Código 2.6 – Busca com uso de expressão regular

```
# output is sorted by status
testsuccessequal "bar/testing 2.0 i386
  $DESCR2

foo/unstable 1.0 all
  $DESCR

package/stable 1.5 all
  $DESCR3

baz/now 0.1 all [installed,local]
```

```
12  $DESCR4
13 " apt search --order-by status -qq "\\w"
```

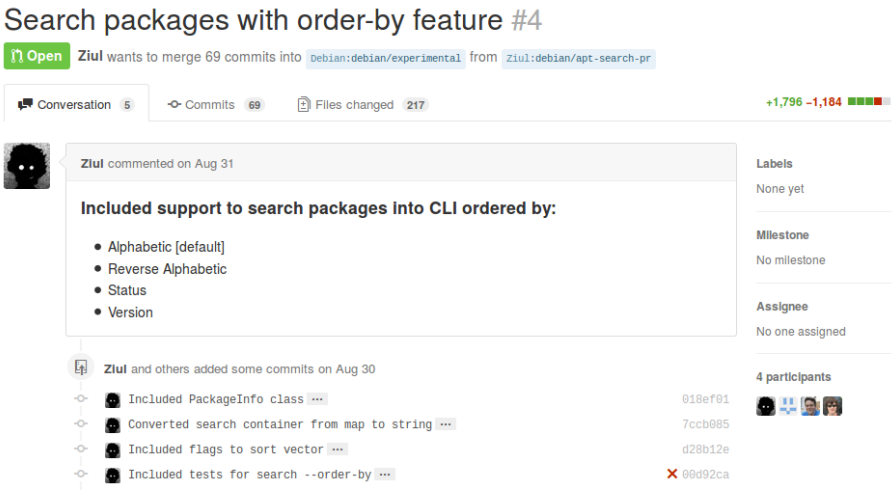

Figura 6 – Submissão do primeiro *merge request*

A Figura 6 apresenta o resultado do *merge request* e alguns dos *commits* enviados, sendo que o último deles traz o resultado do *Travis CI* apontando um problema na *build* devido à falha em um dos testes de integração que verificam concorrência de download de arquivos.

2.2.3 Segunda contribuição

O suporte a expressões regulares amplia a forma como a busca por pacotes pode ser efetuada, possibilitando uma maior gama de opções para desenvolvedores experientes. Todavia, deixar essa opção como padrão acarretaria em alguns pontos negativos:

Alto consumo de memória: Buscas com expressões regulares consomem cerca de $\Theta(2^n)$ de memória para uma expressão de tamanho n, ou $\Theta(n^2)$ para melhores casos, com hardware dedicado (SIDHU; PRASANNA, 2001).

Baixo desempenho: Buscas com expressões regulares tem complexidade de execução $\Theta(n)$, sendo n o tamanho da palavra a ser analisada.

Mesmo a busca com expressões regulares oferecendo um grande poder de busca, questiona-se se o custo que ela requer é aceitável. Vale lembrar que este trabalho visa ampliar a comodidade dos novos usuário Linux e as expressões regulares normalmente não são de conhecimento do usuário principiante. Considerando que este usuário não faria uso comumente, das funções `less`, `more` ou `grep` para refinar as buscas, o uso de expressões regulares não será ampliado.

Procurando por uma alternativa para a busca de pacotes que ofereça menor tempo de processamento e maior economia de memória, foram estudados dois algoritmos de *string matching* exato apresentados na Seção 1.3; o Rabin-Karp e Knuth–Morris–Pratt (KMP). Para ambos os algoritmos foram criadas *branchs* para seu desenvolvimento e testes de desempenho, fazendo de uso da rotina apresentada no Código B.2 para mensurar o tempo gasto para na execução de cada um deles.

Diante dos resultados de tempo de execução obtidos com o algoritmo de *Rabin-Karp*, a segunda contribuição foi planejada e executada. A Seção C.3 apresenta na integra o *diff* da contribuição submetida para o repositório oficial.

A fim de manter a cobertura de teste das funcionalidades, testes foram elaborados para garantir que as buscas continuariam a ser realizadas com ou sem o uso da *flag* que habilitava o uso de expressões regulares, como pode ser observado no Código 2.7, onde um trecho dos testes da segunda submissão é apresentado.

Código 2.7 – Teste com e sem o uso de expressões regulares

```
# output is sorted by status
testequal "bar/testing 2.0 i386
  $DESCR2

foo/unstable 1.0 all
  $DESCR

package/stable 1.5 all
  $DESCR3

baz/now 0.1 all [installed,local]
  $DESCR4
" apt search --order-by status -qq "\\w" --regex

# output is sorted by Alphabetic (non case sense)
testequal "bar/testing 2.0 i386
  $DESCR2

baz/now 0.1 all [installed,local]
  $DESCR4
" apt search --order-by Alphabetic -qq ba
```

Finalizados os testes e a documentação das novas funcionalidades implementadas, foi realizado a submissão de *merge request* ao repositório oficial do APT, a qual pode ser observada na Figura 7 e acompanhada em <https://github.com/Debian/apt/pull/8>.

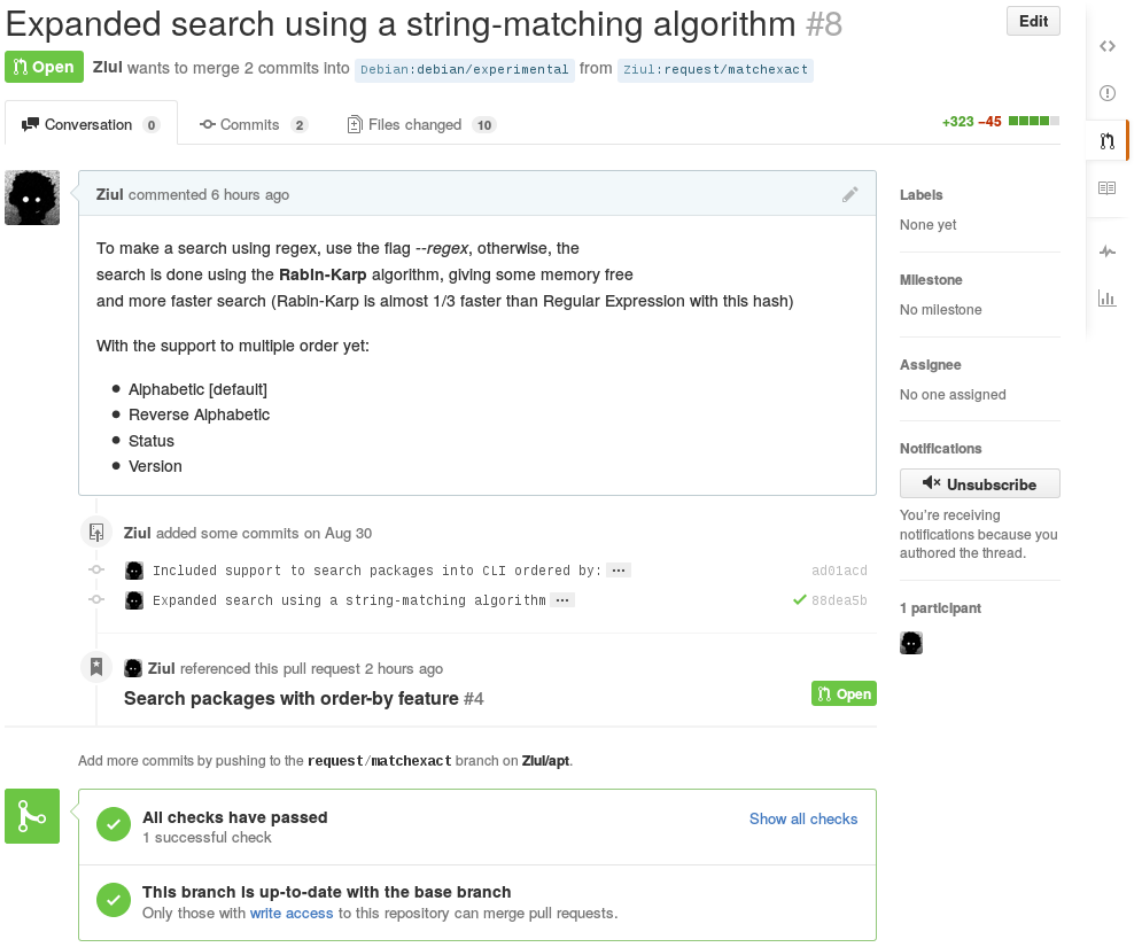

Figura 7 – Submissão do segundo *merge request*

2.2.4 Terceira contribuição

Esta contribuição corresponde ao objetivo principal deste trabalho: oferecer uma ferramenta de busca de pacotes o mais transparente possível quanto ao método de busca. Para atingir tal meta foram realizadas duas tarefas:

Escolha entre expressões regulares ou não: Mesmo oferecendo uma *flag* para selecionar a busca com suporte a expressões regulares, era importante tornar o programa capaz de identificar se o parâmetro passado para a busca envolvia ou não uma *regex* de fato. Isto é, se haviam caracteres especiais ou não no parâmetro de busca.

Inserir método de busca inexata: Quando uma busca não obtiver nenhum resultado, a aplicação deveria executar a busca novamente utilizando um método de busca inexata e apontar os pacotes que poderiam estar relacionados ao parâmetro informado pelo usuário.

Para a primeira tarefa foi implementada uma função que verificava se, entre os parâmetros passados na busca, há caracteres especiais utilizados em expressões regulares.

Estes caracteres são `|\{}[]():*+-$?.^`, porém os caracteres `.`, `:` e `-` costumam aparecer nos nomes dos pacotes os mesmos deveriam ser ignorados como indicativo de uso de expressões regulares.

Código 2.8 – Identificação de Expressões Regulares

```
1  bool identify_regex(std::vector<std::string> input)
2  {
3     /*
4        not all characters can be included, as have
5        packages with the chars .-:
6     */
7     std::string reserver_regex = "^$*+?()[]{}\\|";
8
9     for(auto k:input)
10       if( reserver_regex.find(k) == std::string::npos )
11      return true;
12    return false;
13
14 }
```

O Código 2.8 apresenta uma solução para a identificação de caracteres que indicam o uso de expressões regulares nas buscas, verificando cada palavra em uma lista de *strings* em busca dos caracteres citadas anteriormente. Caso algum caractere especial seja localizado, a função retorna verdadeiro.

Em seguida, foi verificada a *flag* de uso de expressão regular ou caracteres reservados com o seguinte trecho de código:

```
if ((_config->FindB("APT::Cache::UsingRegex",false)) || identify_regex(args))
```

Caso a chamada acima seja verdadeira, as rotinas que compilam as expressões regulares serão executadas; caso contrário, o método de Rabin-Karp será executado para realizar a busca, visto este método ter apresentado melhor desempenho em relação ao método de Knuth–Morris–Pratt (KMP), dado que será melhor apresentando em maiores detalhes na Seção 3.2.

Finalizada esta etapa, o próximo, e último, passo foi identificar buscas onde nenhum pacote era localizado. Nestes casos, o algoritmo de *Leveinstein* deveria ser utilizado para encontrar os pacotes que tivessem a menor diferença com o texto da pesquisa realizada. Essa chamada deveria ser realizada somente nos casos onde não houvesse o uso de expressões regulares.

Para realizar uma busca utilizando o modelo de *Leveinstein* era necessário percorrer a lista de pacotes duas vezes. Na primeira vez eram pontuadas as distâncias dos nomes dos pacotes com os parâmetros de entrada da busca; na segunda vez os pacotes

eram ordenados de acordo com sua pontuação. O Código 2.9 apresenta a realização desta busca, percorrendo toda a lista de pacotes disponível (`bag`), e em seguida ordenando a saída.

Código 2.9 – Busca ordenada por distância de *Levenshtein*

```
for(auto V:bag)
{

    // we want to list each package only once
    pkgCache::PkgIterator const P = V.ParentPkg();
    if (PkgsDone[P->ID] == true)
        continue;

    std::string PkgName = P.Name();
    pkgCache::DescIterator Desc = V.TranslatedDescription();
    pkgRecords::Parser &parser = records.Lookup(Desc.FileList());
    std::string const LongDesc = parser.LongDesc();

    for (auto arg:args)
    {
        int distance = levenshtein_distance(PkgName, arg);
        if ((distance >=0) && (distance <= (int)PkgName.length()/2 ))
        {
            PkgsDone[P->ID] = true;
            std::stringstream outs;
            ListSingleVersion(CacheFile, records, V, outs, format);
            outputVector.emplace_back(CacheFile, records, V, outs.str());
            outputVector.back().distance = distance;
        }
    }
}
std::sort(outputVector.begin(), outputVector.end(), OrderByDistance);
```

Devido ao fato das buscas inexatas consumirem mais recursos e de passarem duas vezes pela lista dos pacotes, elas foram delegadas apenas aos casos onde a busca não obtivesse nenhum resultado. O Código 2.10 ilustra esta situação.

Código 2.10 – Chamada seletiva para buscas inexatas

```
if(outputVector.size() == 0)
{
    outputVector = SimilarSearch(bag,args);
    if(outputVector.size() > 10)
    outputVector.resize(10);
```

```
6     std::cout << "None package fould. Maybe you are looking for one of those:" <<
          std::endl;
7 }
```

Para o desenvolvimento desta contribuição, foi aproveitada a contribuição anterior, apenas atualizando a *branch* com os novos *commits*. Na Figura 8 podemos observar o mesmo *merge request* requisitado para a segunda contribuição agora com o acréscimo do *commit* `aae1adc`, onde todas as contribuições do terceiro grupo de contribuições deste trabalho foram agrupadas para simplificar a visualização e revisão do *merge request*.

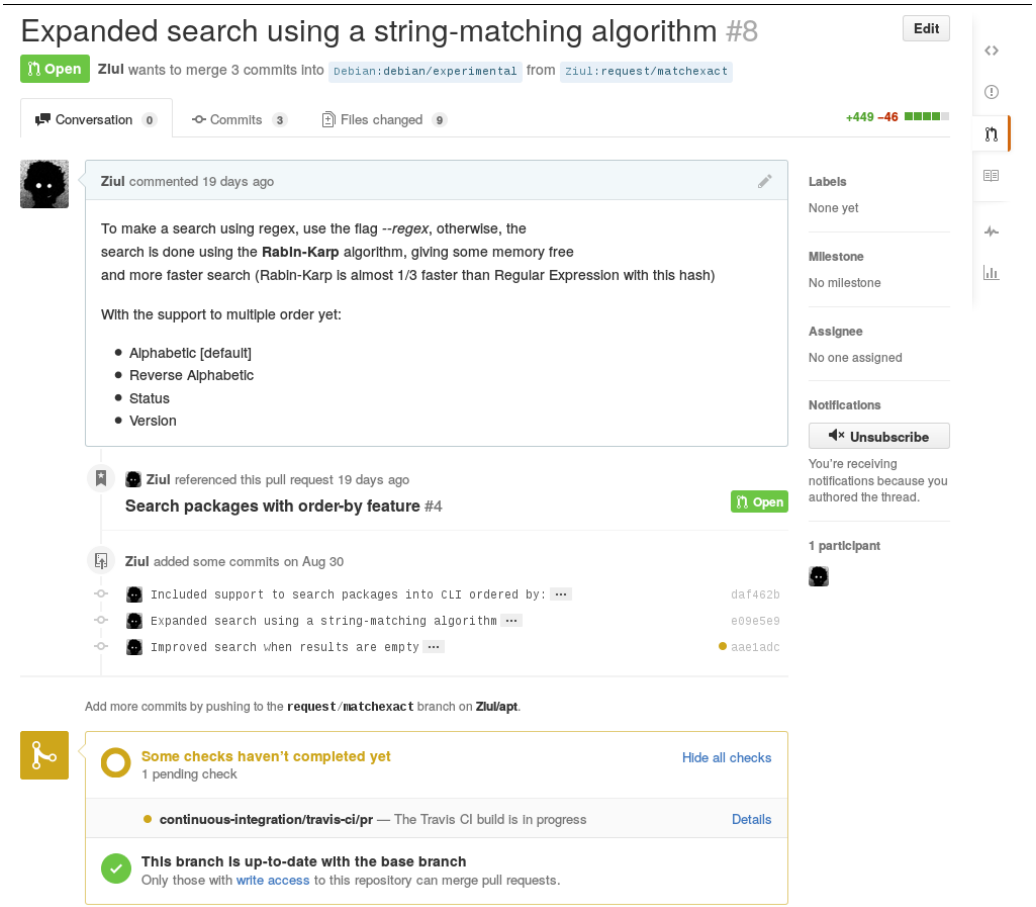

Figura 8 – Submissão do segundo *merge request* atualizado com o conteúdo da terceira contribuição

Parte II

Resultados

3 Resultados e Discussão

Neste capitulo serão analisados os resultados obtidos no capitulo anterior, que servirão de motivo para a tomada de decisão de como a contribuição será realizada. Resultados como desempenho, consumo de memória e custos de implementação serão apontadas com o objetivo de comparação entre os algoritmos.

3.1 Contribuições

Para a realização deste trabalho foram feitos contatos com os desenvolvedores oficiais do `APT` por duas vias distintas. Na primeira contribuição, o *pull request* foi comprometido devido ao *Travis CI* apontar a quebra de três testes que já estavam no repositório. A primeira forma de comunicação então foi via email com o responsável por essa contribuição, explicando que havia uma submissão, porém ela provavelmente seria ignorada devido ao teste falho. A resposta do email explicava que alguns testes de integração do `APT` realizam concorrência e podem vir a falhar, assim como um teste especifico que tentava baixar o mesmo pacote duas vezes para comparar seu tamanho, porém o *Travis CI* eventualmente limitava a banda da seção, o que resultava na falha do teste por perca de conexão. Pouco depois já haviam os primeiros comentários sobre o primeiro *pull request* pelo GitHub. Porém ao final dos comentários, sugestões e correções do deste *pull request*, houve um pedido de confirmação ao dono do repositório se o comentário no código original apontando um pedido de correção ainda tinha necessidade e se a submissão daquela contribuição poderia ser aprovada. Infelizmente, ate a data de entrega deste trabalho não havia recebido resposta para essa pergunta do dono do repositório.

Como consequência, há um *pull request* aberto com três *commits* esperando uma revisão contendo as três contribuições programadas para este trabalho. Ate a entrega deste trabalho, não havia qualquer comentário nas contribuições.

3.2 Algoritmos de *string matching* exatos

Para a busca com algoritmos de *string matching* exatos, foram considerado três abordagens distintas:

Expressão Regular: Método atual de verificação. Foi comentado previamente na Seção 1.3.

Knuth-Morris-Pratt: Algoritmo que faz de uso do conceito de autômatos de estados para acelerar a busca. Previamente apresentado na Seção 1.3.1.

Rabin-Karp: Algoritmo que faz de uso de *hash*. Ver Seção 1.3.1.

Após a coleta de 150 execuções de busca para dez pacotes usando os algoritmos apontados acima, a Seção 1.3.1 foi elaborada a partir da a mediana dos tempos destas execuções.

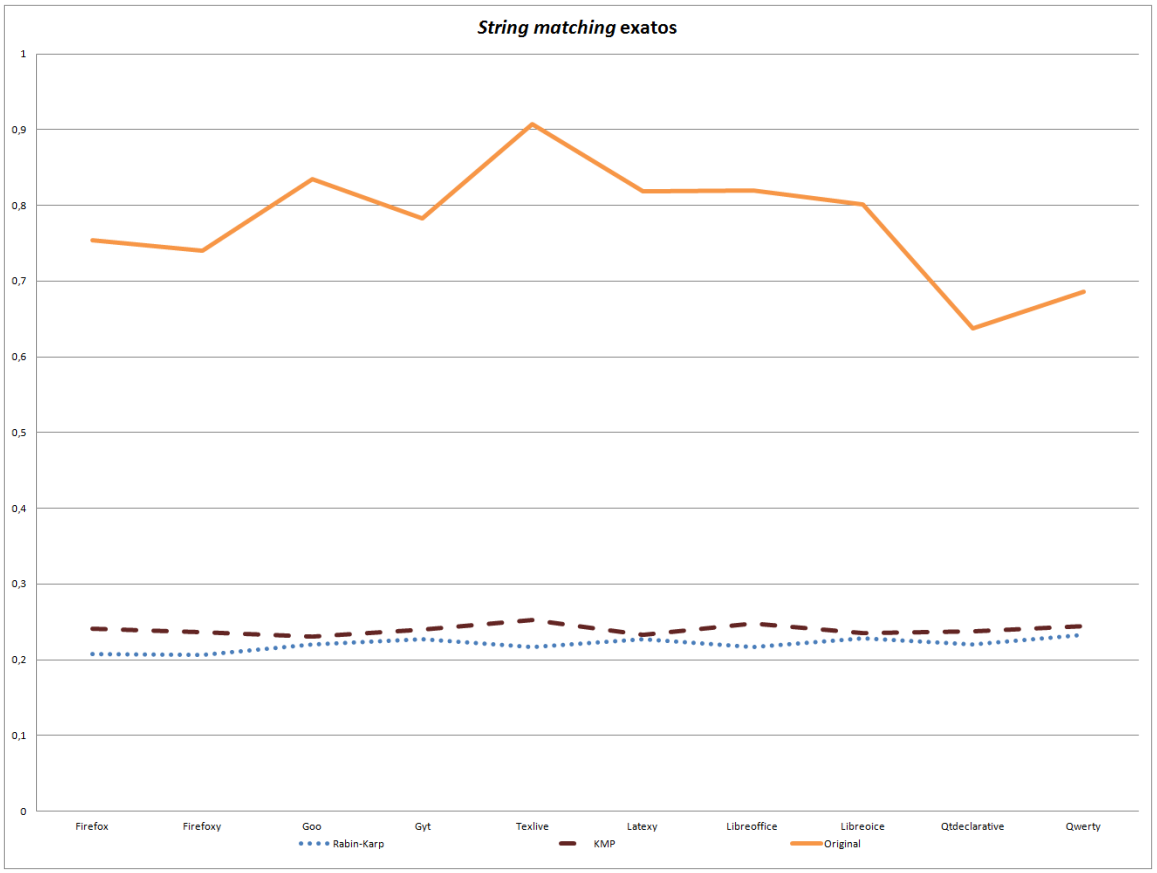

Figura 9 – Estimativa de tempo para pacotes usando algoritmos de busca exata

Como podemos observar, tanto o *Rabin-Karp* quanto o *KMP* tiveram um tempo de execução de cerca de $\frac{1}{3}$ do tempo gasto atualmente com o uso de expressões regulares, sendo que, no geral, o algoritmo de *Rabin-Karp* possui um desempenho ligeiramente melhor.

O consumo total de memória para ambos os algoritmos, *Rabin-Karp* na Figura 10 e expressões regulares na Figura 11, apresentaram resultados similares, porém o método de *Rabin-Karp* faz um uso de memória mais pontual, alcançando o pico de consumo ao final do processo, quando esta prestes a liberar os recursos. Já o método de expressões regulares apresenta um consumo aproximadamente linear de memória.

Um gasto mais prologado de memória vem a ser prejudicial para sistemas em que diversos processos possam estar sendo executados em conjunto; Todavia, o grau de consumo é baixo, evitando que este consumo por um período mais extenso venha a ser

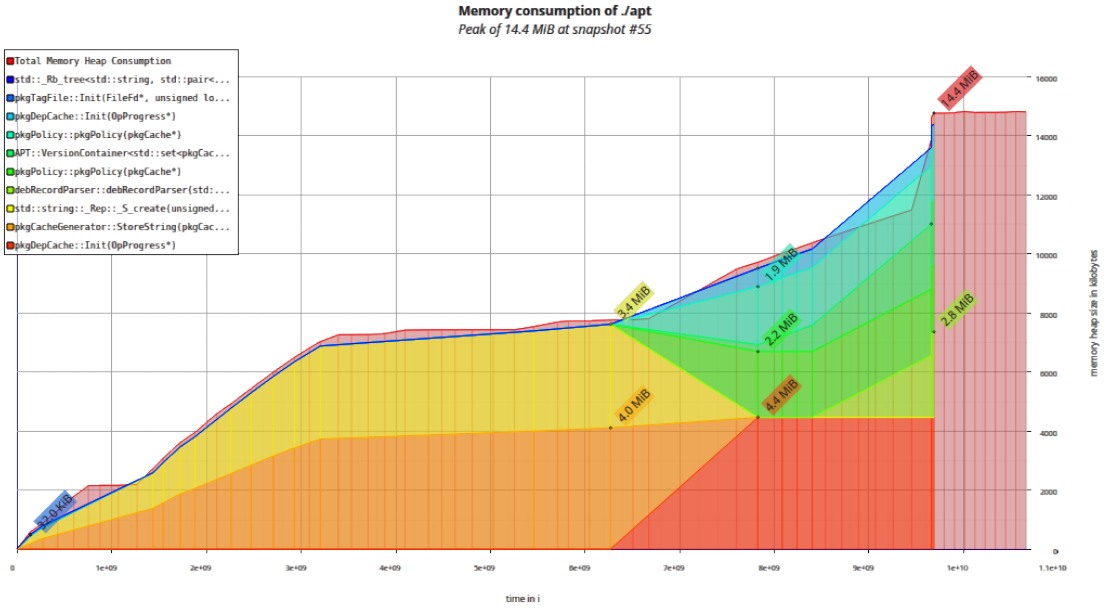

Figura 10 – Uso de memória com uso do algoritmo de *Rabin-Karp*

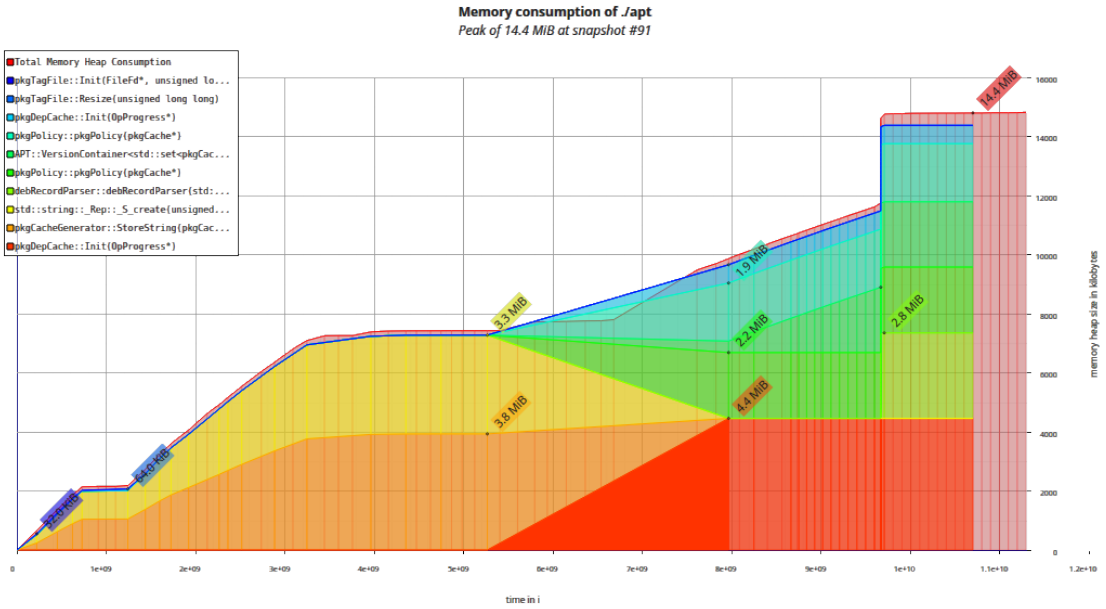

Figura 11 – Uso de memória das expressões regulares

prejudicial. Em um sistema com 2GB de memória RAM, os 15MB utilizados representam menos de 1% do total dos recursos disponíveis.

3.3 Algoritmos de *string matching* inexatos

Para a busca com métodos de *string matching* inexatos, foram considerado dois algoritmos distintos:

Levenshtein: Algoritmos voltado para correção de erros. Foi previamente comentado na Seção 1.3.2.

Coeficiente de Sørensen–Dice: Algoritmos que faz de uso do conceito de autômatos de estados para acelerar a busca. Previamente apresentado na Seção 1.3.2.

A tomada desta decisão foi baseada nos estudos realizados previamente à este trabalho, onde havia-se considerado três modos distintos de busca no intuito de prototipar uma solução utilizando o *bind* em Python do `APT`. Para validação dos protótipos, 7 (sete) dentre os 30 (trinta) pacotes mais instalados segundo o Popcorn-Ubuntu[1] foram escolhidos para realizar os testes de busca com cada um destes e os protótipos.

Para efeito de comparação, os testes foram realizados também com o `apt` e o `apt-cache`. Os tempos obtidos foram apresentados na Tabela 4. Para uma melhor apresentação dos dados, buscas com mais de 100 resultados foram desconsiderados e identificadas no final da tabela. As opções de uso de *pool de threads* não foram utilizadas devido a sua instabilidade no momento de ordenação dos resultados. A Figura 12 apresenta um gráfico representativo da Tabela 4 com a supressão dos valores acima de 100 para melhor visualização.

Os dados apresentados na Tabela 4 indicam uma certa deficiência em apresentar os pacotes realmente desejados no topo da lista de possibilidades quando se trata dos métodos padrões de busca, `apt` e `apt-cache`. Com a busca realizada no `apt`, a busca pelo pacote `bash` foi a mais bem sucedida, posicionando o pacote desejado na 6ª posição, porém teve resultados terríveis quando observado a busca pelos pacotes `sed` e `tar`, onde ambos apareciam na lista, após a 10.000ª posição

Devido o formato como os protótipos de *Levenshtein* e *Smith* estão escritos, é possível se imprimir uma comparação com toda a lista de pacotes disponíveis na *cache*. Assim, apenas os 50 primeiros pacotes eram impressos para verificação. Desta forma, a linha *dimensão* da tabela para estes algoritmos foi desconsiderada.

O que podiamos observar com os protótipos é um resultado proporcional ao que se esperava, de quanto maior o trabalho ou a complexidade do algoritmo para a comparação

[1] Disponível em <http://popcon.ubuntu.com/>.

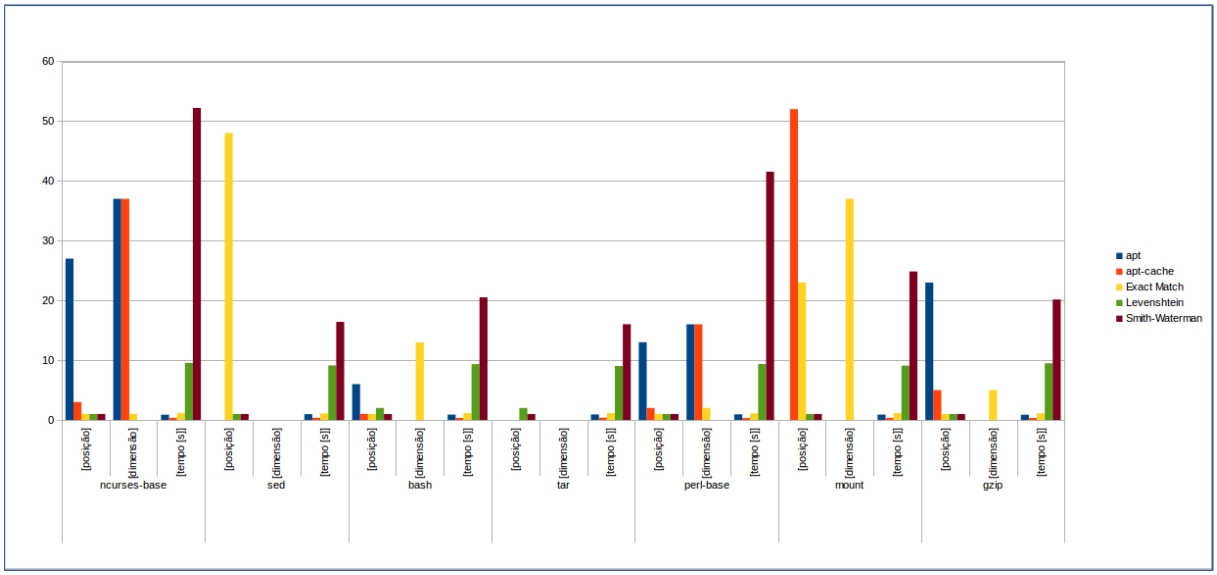

Figura 12 – Amostra de comparação dos resultados

Tabela 4 – Comparação de resultados

Pacote		apt	apt-cache	Exact Match	Levenshtein	Smith-Waterman
	posição	27	3	1	1	1
ncurses-base	*dimensão*	37	37	1	–	–
	tempo [s]	0.874	0.349	1.146	9.539	52.212
	posição	***	***	48	1	1
sed	*dimensão*	***	***	*	–	–
	tempo [s]	0.968	0.358	1.102	9.125	16.421
	posição	6	1	1	2	1
bash	*dimensão*	*	*	13	–	–
	tempo [s]	0.901	0.335	1.142	9.343	20.518
	posição	***	*	*	2	1
tar	*dimensão*	*	*	*	–	–
	tempo [s]	0.923	0.373	1.137	9.028	16.02
	posição	13	2	1	1	1
perl-base	*dimensão*	16	16	2	–	–
	tempo [s]	0.936	0.311	1.116	9.36	41.548
	posição	*	52	23	1	1
mount	*dimensão*	*	*	37	–	–
	tempo [s]	0.895	0.338	1.121	9.082	24.848
	posição	23	5	1	1	1
gzip	*dimensão*	*	*	5	–	–
	tempo [s]	0.876	0.324	1.103	9.482	20.163

***** Acima de 100 pacotes.

****** Acima de 1000 pacotes.

******* Acima de 10000 pacotes.

– Quantidade desconsiderada

de *strings* e qualificação dos resultados, maior será o tempo necessário para a pesquisa, porém melhor serão os resultados obtidos na qualificação.

Uma situação ideal seria obter os resultados alcançados com o algoritmo de *Smith-Waterman* com o tempo de resposta do `apt-cache`. Porém a realidade é que o tempo gasto pelo algoritmo de *Smith-Waterman* é impraticável, visto que quanto maior o tamanho do nome do pacote, maior é o tempo de resposta, levando a gastar quase 1 (um) minuto para uma busca pelo pacote `ncurses-base`. Porém o tempo gasto pelo algoritmo de *Levenshtein* se mostrava válido para futuros estudos de otimização e inclusão nas rotinas de busca quando retornam poucos pacotes ou nenhum pacote, considerando assim que possa ter acontecido um erro de digitação. Neste caso seria importante apresentar ao usuário que existe a possibilidade de escrita incorreta de pacote antes da apresentação dos resultados.

Tabela 5 – Comparação de resultados com entradas contendo erros de ortografia

Pacote		**apt**	**apt-cache**	**Exact Match**	**Levenshtein**	**Smith-Waterman**
pitom-dev	*posição*	0	0	0	15	11
[**python-dev**]	*dimensão*	0	0	0	–	–
	tempo [s]	0.894	0.325	1.1	9.443	45.148
gyth-core	*posição*	0	0	0	1	16
[**git-core**]	*dimensão*	0	0	0	–	–
	tempo [s]	0.872	0.316	1.124	9.757	40.694
libbesh	*posição*	0	0	0	4	2
[**libbash**]	*dimensão*	0	0	0	–	–
	tempo [s]	0.816	0.331	1.121	9.481	32.686
g-zip	*posição*	0	0	0	1	1
[**gzip**]	*dimensão*	0	0	0	–	–
	tempo [s]	0.881	0.312	1.118	9.237	24.224

– Quantidade desconsiderada

Também foi verificado uma busca apenas com a escrita errada de pacotes para simular o desconhecimento do nome do pacote pelo usuário, fazendo com que fosse influenciado apenas pela fonética do pacote, tanto para simular um casual erro de digitação que se passa despercebido. A Tabela 5 apresenta estes resultados para alguns pacotes - contendo entre colchetes o nome correto do pacote. Como já é de conhecimento, as aplicações `apt` e `apt-cache` não possuem algoritmo que possa buscar por pacotes com o nome semelhante ao digitado quando não há resultados. Desta forma, as saídas para estas aplicações foram nulas devido a não existir um pacote com o nome fornecido na busca. Por fazer um processo de busca exata, o algoritmo de nome exato também não conseguiu encontrar nenhum pacote para apresentar de resultado quando era fornecido um nome inexistente dentre os pacotes disponíveis.

Para os algoritmos de *Levenshtein* e *Smith-Waterman* os resultados já são bem diferentes, já que ambos apresentam resultados baseados em aproximação de nomes. O algoritmo de *Levenshtein* conseguiu manter três dos quatro testes entre os cinco primeiros

resultados. Já o de *Smith-Waterman* conseguiu manter apenas em metade dos casos o pacote desejado entre os cinco primeiros - mesmo tendo um melhor resultado na procura pelo pacote `pithom-dev` onde foram inseridos dois erros ortográficos.

Posteriormente, o Coeficiente de Sørensen–Dice foi considerado uma possível solução em comparativo com o algoritmo de *Levenshtein* por ter ordem de complexidade semelhante ao das expressões regulares, já sendo utilizado na etapa de implementação em C++, pulando assim a fase de prototipação, como o algoritmo de *Levenshtein* e *Smith-Waterman* tiveram.

Para análise de desempenho, foram coletadas 150 amostras de tempo para 10 buscas por pacotes distintos, das quais metade eram de pacotes que não existiam. A Figura 13 apresenta a mediana destes tempos.

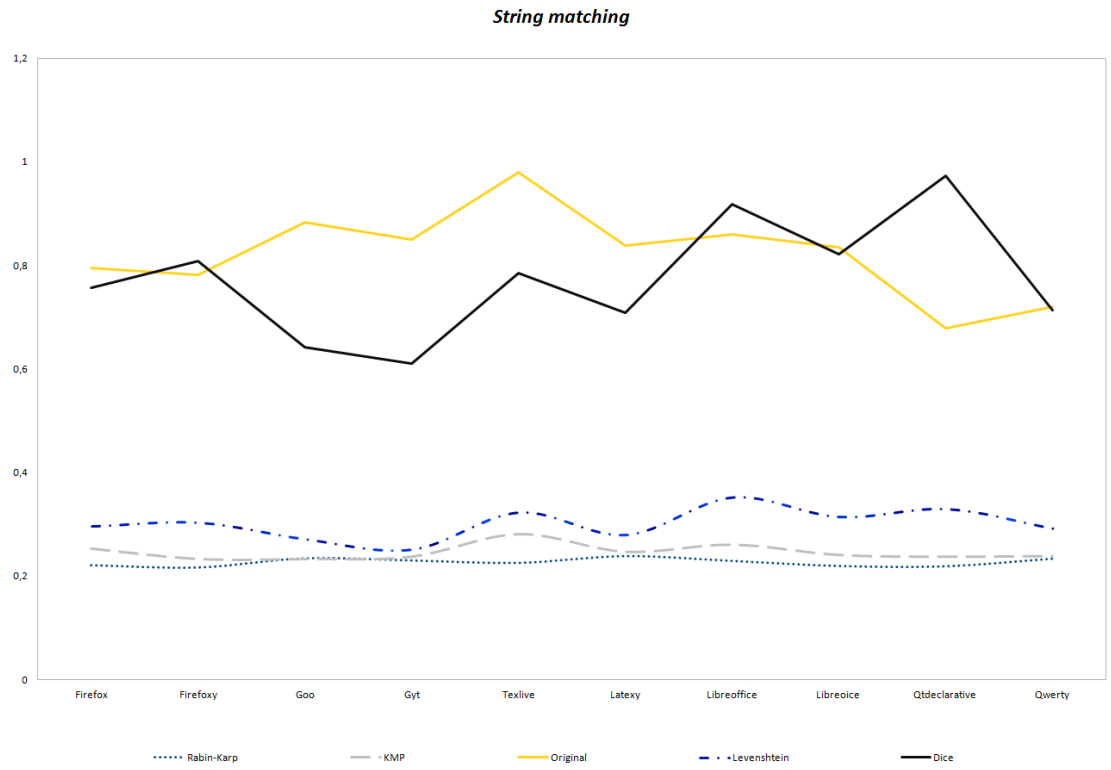

Figura 13 – Estimativa de tempo para pacotes usando algoritmos de busca inexata

Tabela 6 – Tempo estimado para busca por pacote com cada algoritmo.

Pacote/Tempo [s]	Rabin-Karp	KMP	Original	Levenshtein	Dice
Firefox	0.220749	0.252704	0.795402	0.296975	0.756102
Firefoxy	0.216384	0.232433	0.781483	0.304191	0.808109
Goo	0.234158	0.232317	0.882526	0.272290	0.641333
Gyt	0.230002	0.236883	0.850554	0.252317	0.611070
Texlive	0.225326	0.281123	0.979610	0.323858	0.784428

Latexy	0.238537	0.246641	0.837671	0.280488	0.708014
Libreoffice	0.229116	0.260430	0.860675	0.352967	0.918178
Libreoice	0.219333	0.240316	0.834481	0.315241	0.821371
Qtdeclarative	0.218691	0.236788	0.678371	0.330715	0.973538
Qwerty	0.233489	0.238262	0.720777	0.292980	0.713505

Por fim, pode-se afirmar que dentro dos algoritmos selecionados para este trabalho, o algoritmo de *Leveinstein* apresenta um dos melhores tempos para buscas inexatas de *strings*. A Figura 13 reforça essa afirmação, ao mostrar que o tempo de resposta deste algoritmo é inferior ao tempo gasto para buscas com expressões regulares e ficando pouco acima do tempo necessário para realizar buscas exatas com o algoritmo de *Rabin-Karp*.

4 Considerações Finais

O uso dos algoritmos selecionados para substituir o uso de expressões regulares apresentaram ganhos de desempenho notáveis, oferecendo buscas que fazem menos uso de recursos e apresentam os resultados em menor tempo; Entretanto, o us de memória não mostrou alterações significativas.

O grupo responsável pela manutenção e evolução do `APT` segue um guia de estilo de código extremamente peculiar, apresentado no Código B.1. De acordo com o guia de estilo seguido, as indentações devem ser realizadas com espaço de três para cada nível, porém a cada oito espaços, os mesmos devem ser substituídos por tabulações.

Um hábito comum com o desenvolvimento do `APT` é o fato de manter *builds* em que o *status* na aplicação *Travis CI* esta quebrada, devido ao fato de grande parte dos testes serem realizados com chamadas concorrentes. Devido ao tempo e ordem destas chamadas não serem determinísticas, muitas vezes um teste falha por estar definido uma ordem diferente do resultado obtido pela chamada paralela. Este mal habito foi um dos motivos pelos quais múltiplas contribuições foram e vem sendo recusadas, visto que o GitHub aponta a *build* como quebrada e recomenda a recusa. Outra falha comum nos testes é um cenário onde é necessário adquirir o mesmo arquivo múltiplas vezes. Este teste falha devido a instabilidade da plataforma de integração continua (*Travis CI*), que muitas das vezes resulta em um *timeout* no *download* do arquivo, gerando a falha no teste.

Para um software de grande responsabilidade como o APT, esperava-se classes pequenas com métodos objetivos e coesos, porém é comum observar métodos e funções com mais de trinta linhas com múltiplas obrigações. Como consequência, há funções com mesmos objetivos ou objetivos semelhantes, provavelmente devido a dificuldade em se encontrar um método dedicado para uma única tarefa.

Mesmo não aceitas ainda até a entrega do trabalho, as contribuições ofereciam melhorias para os usuários que desejavam poder ordenar a listagem de algum pacote não apenas de acordo com a ordem alfabética, oferecendo mais liberdade. Além de que, buscas com resultados vazios não existem mais, já que esta situação passa a ser considerada como um possível erro ortográfico do usuário e os pacotes com escrita mais próxima do escrito são apresentados.

Para o autor, trabalhar com um software livre de tamanho impacto como o `APT` ofereceu experiencias que não poderiam ser alcançadas facilmente com projetos pequenos. A necessidade de manter a compatibilidade com diversas arquiteturas, a taxa de cobertura de código não apenas alta, mas por vezes extensa na tentativa de simular todas as possíveis combinações de entradas para o programa. A necessidade de que uma funcionalidade

não pode ser alterada se sua documentação não for atualizada também demonstra a preocupação dos desenvolvedores em garantir que o usuário final terá meios de conhecer melhor a ferramenta a fim de usufruir de todos os seus recursos. Ou a fidelidade ao guia de estilo de desenvolvimento, que por mais que pareça sem sentido, a equipe respeita e preserva. Porém pontos ruins foram observados como a forma que o grupo se comporta, se mantendo fechado a novas opções, seja por ser contrario a elas, ou simplesmente por ter a necessidade de manter o padrão onde a decisão deve ser tomada em grupo e só quando há empate de opiniões ou duvidas sobre determinada funcionalidade que o criador da ferramenta pode utilizar sua influencia para a tomada de decisões.

Trabalhos Futuros

Apesar de ser um programa maduro e com alta confiabilidade pelos usuário, ainda existem diversas formas de contribuir com o `APT`. A contribuição mais simples esta justamente nas traduções, visto que muito das funcionalidades ainda não foram traduzidas. Porém há espaço para contribuir desenvolvendo código do `ATP`. Em diversos pontos do programa pode-se ver um acoplamento muito grande de métodos e funções, com arquivos com mais de 300 linhas contendo apenas uma função, ou a duplicação de rotinas devido a má estruturação dos arquivos e funções. Na própria busca por pacotes podemos observar uma duplicidade de afazeres quando a requisição à *cache* de pacotes gera uma ordenação antes do envio dos pacotes, para uma função que irá reordenar os pacotes antes de imprimir a mensagem.

Referências

APT-RPM. *apt-rpm.org*. 2010. Disponível em: <http://apt-rpm.org/about.shtml>. Acesso em: 15 de dezembro de 2015. Citado na página 33.

BAILEY, E. C. *Maximum RPM: Taking the red hat package manager to the limit*. [S.l.]: Red Hat Software, 1997. Citado na página 30.

BECK, M.; MAGNUS, R.; KUNITZ, U. *Linux Kernel Internals with Cdrom*. [S.l.]: Addison-Wesley Longman Publishing Co., Inc., 2002. Citado na página 29.

BORTZMEYER, S. *dselect Documentation for Beginners*. Debian Documentation Project, 1999. Disponível em: <https://www.debian.org/doc/manuals/dselect-beginner/>. Acesso em: 24 de novembro de 2015. Citado na página 32.

BRETTHAUER, D. Open source software in libraries. *Library Hi Tech News*, Emerald Group Publishing Limited, v. 18, n. 5, 2001. Citado na página 29.

CAVNAR, W. B.; TRENKLE, J. M. et al. N-gram-based text categorization. *Ann Arbor MI*, Citeseer, v. 48113, n. 2, p. 161–175, 1994. Citado na página 40.

DAMERAU, F. J. A technique for computer detection and correction of spelling errors. *Communications of the ACM*, ACM, v. 7, n. 3, p. 171–176, 1964. Citado na página 38.

DEBIAN GNU/LINUX. *The Debian GNU/Linux FAQ*. [S.l.], 2013. Citado na página 30.

DICE, L. R. Measures of the amount of ecologic association between species. *Ecology*, JSTOR, v. 26, n. 3, p. 297–302, 1945. Citado na página 40.

FEOFILOFF, P. *Projeto de Algoritmos*. 2015. Disponível em: <www.ime.usp.br/~pf/algoritmos/idx.html>. Acesso em: 4 de outubro de 2015. Citado 2 vezes nas páginas 34 e 35.

GARBEE, B. et al. A brief history of debian. *the package: Debian-history*, 2008. Citado na página 32.

HEKMAN, J. P.; ORAM, A. *Linux in a Nutshell*. [S.l.]: O'Reilly & Associates, Inc., 1996. Citado na página 31.

HUME, A.; SUNDAY, D. Fast string searching. *Software: Practice and Experience*, Wiley Online Library, v. 21, n. 11, p. 1221–1248, 1991. Citado na página 36.

KNUTH, D. E. *The art of computer programming: sorting and searching*. [S.l.]: Pearson Education, 1998. Citado na página 35.

LEVENSHTEIN, V. I. Binary codes with correction of drooping and insertion of the symbol 1. In: *Problemy Peredachi Informatsii*. [S.l.: s.n.], 1965. v. 1, p. 12–25. Citado 2 vezes nas páginas 36 e 38.

LEVENSHTEIN, V. I. Binary codes capable of correcting deletions, insertions and reversals. In: *Soviet physics doklady*. [S.l.: s.n.], 1966. v. 10, p. 707. Citado 2 vezes nas páginas 36 e 37.

NEEDLEMAN, S. B.; WUNSCH, C. D. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology*, Elsevier, v. 48, n. 3, p. 443–453, 1970. Citado na página 38.

SHIREY, R. *RFC 4949–Internet Security Glossary*. [S.l.]: Version, 2007. Citado na página 37.

SIDHU, R.; PRASANNA, V. K. Fast regular expression matching using fpgas. In: IEEE. *Field-Programmable Custom Computing Machines, 2001. FCCM'01. The 9th Annual IEEE Symposium on*. [S.l.], 2001. p. 227–238. Citado na página 54.

SMITH, T. F.; WATERMAN, M. S. Identification of common molecular subsequences. *Journal of molecular biology*, Elsevier, v. 147, n. 1, p. 195–197, 1981. Citado 3 vezes nas páginas 38, 40 e 41.

SØRENSEN, T. {A method of establishing groups of equal amplitude in plant sociology based on similarity of species and its application to analyses of the vegetation on Danish commons}. *Biol. Skr.*, v. 5, p. 1–34, 1948. Citado na página 40.

THIERBACH, M. E. *Finite state machine*. [S.l.]: Google Patents, 1985. US Patent 4,493,029. Citado na página 35.

Apêndices

APÊNDICE A – Protótipos elaborados no trabalho

Código A.1 – Protótipo em *Python* para estudo do pacote `apt`.

```
# -*- coding: utf-8 -*-
"""
Prototype code

Prototype code to teste apk API
"""


def test_install(package_name="git"):
    """
    function to test a package installation
    """
    try:
        import apt
        from apt import progress
    except ImportError as error:
        raise error

    cache = apt.cache.Cache(progress.text.OpProgress())

    try:
        cache.update(progress.base.AcquireProgress())
        cache.open(progress.text.OpProgress())
        pkg = cache[package_name]
    except KeyError:
        print "{pkg_name} not found".format(pkg_name=package_name)
        return
    except apt.cache.LockFailedException as arg:
        print "Error: {err}".format(err=str(arg))
        return

    if pkg.is_installed:
        print "{pkg_name} already installed".format(pkg_name=package_name)
    else:
        # select to be installed
        pkg.mark_install()

```

```
38        print "Dependeces from %s to be installed:" % package_name
39        print cache.get_changes()[0].name,
40        for i in cache.get_changes():
41            print i.name,
42
43        if raw_input("\nDo you want to continue? (y/n) ") == 'y':
44            try:
45                cache.commit(install_progress=progress.base.InstallProgress())
46            except apt.cache.LockFailedException as arg:
47                print "Error: {err}".format(err=str(arg))
48        else:
49            cache.clear()
50
51        # library compatibility
52        try:
53            cache.close()
54        except AttributeError as error:
55            pass
56        print "Leaving"
57
58 if __name__ == '__main__':
59     test_install('gito')
```

Código A.2 – Protótipo do algoritmo de *match* exato.

```
1  # -*- coding: utf-8 -*-
2  """
3  Code to rank packages from a search in APT using exact match
4  """
5
6  from apt import Cache
7  import optparse
8  from multiprocessing.pool import ThreadPool as Pool
9  _MAX_PEERS = 20
10
11
12 class Pack(object):
13
14     """Class to handle the package object"""
15     name = ''
16     ratio = 0
17
18     def __init__(self):
19         super(Pack, self).__init__()
20
21     def __str__(self):
```

```
22          return self.name.ljust(50) + ' ' + str(self.ratio)
23
24      def __gt__(self, other):
25          return self.ratio > other.ratio
26
27      def __lt__(self, other):
28          return self.ratio < other.ratio
29
30      def __eq__(self, other):
31          return self.ratio == other.ratio
32
33      def __le__(self, other):
34          return self.ratio <= other.ratio
35
36      def __ge__(self, other):
37          return self.ratio >= other.ratio
38
39      def __ne__(self, other):
40          return self.ratio != other.ratio
41
42      def __hash__(self):
43          # return self.ratio
44          return hash(self.__str__())
45
46      def __len__(self):
47          return len(self.name)
48
49  _parser = optparse.OptionParser(
50      usage="Use with care",
51      description="Search packages"
52  )
53
54  _parser.add_option("--no-suffix",
55                     dest="suffix",
56                     action="store_false",
57                     help="suppres search for suffixes",
58                     default=True
59                     )
60  _parser.add_option("--no-prefix",
61                     dest="prefix",
62                     action="store_false",
63                     help="suppres search for prefix",
64                     default=True
65                     )
66
67  _parser.add_option("--amount",
68                     dest="amount",
```

```
                    type='int',
                    help="How many print",
                    default=50
                    )

_parser.add_option("--ratio",
                    dest="ratio",
                    type='float',
                    help="Minimal ratio to print",
                    default=0.0
                    )

_parser.add_option("--multi",
                    dest="single",
                    action="store_false",
                    help="Depraced",
                    default=True
                    )


def _cmp(pack, other):
    """ Ovewriting for compare method"""
    return pack.name < other.name


def Thread_Exact(k):
    """ Method to be done by a thread from the pool"""
    pack = args[0]
    if pack in k:
        item = Pack()
        item.name = k
        item.ratio = 0
        return item
    return None


def Exact_Match(pack):
    """ Method to execute the exact matching """
    cache = Cache()
    if _options.single:
        list_app = []
        for k in cache:
            if pack in k.name:
                item = Pack()
                item.name = k.name
                item.ratio = 0
                list_app.append(item)
```

```
116         return list_app
117     else:
118         _pool = Pool(processes=_MAX_PEERS)
119         result = _pool.map(Thread_Exact, cache._set)
120         return result
121 
122 if __name__ == '__main__':
123     (_options, args) = _parser.parse_args()
124     package_name = args[0]
125     suffixes = ['core', 'dev', 'commom', 'devel']
126     prefixes = ['lib']
127 
128     _options.suffix = _options.prefix = False
129 
130     lista = Exact_Match(package_name)
131     if _options.suffix:
132         for suffix in suffixes:
133             matches = Exact_Match('{}-{}'.format(package_name, suffix))
134             lista.extend(matches)
135     if _options.prefix:
136         for prefix in prefixes:
137             matches = Exact_Match('{}{}'.format(prefix, package_name))
138             lista.extend(matches)
139     if _options.suffix and _options.prefix:
140         for suffix in suffixes:
141             for prefix in prefixes:
142                 matches = Exact_Match(
143                     '{}{}-{}'.format(prefix, package_name, suffix))
144                 lista.extend(matches)
145 
146     lista = filter(None, lista)
147     lista = sorted(lista, cmp=_cmp)
148     _options.amount = 300
149     for i in lista[:_options.amount]:
150         print i
151 
152     print '%d itens' % len(lista)
```

Código A.3 – Protótipo do algoritmo de *Levenshtein*.

```
1 # -*- coding: utf-8 -*-
2 """
3     Code to rank packages from a search in APT using Levenshtein
4 """
5 from apt import Cache
6 from Levenshtein import ratio
```

```
7  from exact import Pack, _parser
8  from multiprocessing.pool import ThreadPool as Pool
9
10 _MAX_PEERS = 20
11
12
13 def Thread_Rank(k):
14     pack = _args[0]
15     item = Pack()
16     item.name = k
17     item.ratio = ratio(pack, k)
18     return item
19
20
21 def Rankilist(pack):
22     cache = Cache()
23     if _options.single:
24         list_app = []
25         for k in cache:
26             item = Pack()
27             item.name = k.name
28             item.ratio = ratio(pack, k.name)
29             list_app.append(item)
30         return list_app
31     else:
32         _pool = Pool(processes=_MAX_PEERS)
33         result = _pool.map(Thread_Rank, cache._set)
34         return result
35
36 if __name__ == '__main__':
37     (_options, _args) = _parser.parse_args()
38     package_name = _args[0]
39     suffixes = ['core', 'dev', 'commom', 'devel']
40     prefixes = ['lib']
41
42     lista = Rankilist(package_name)
43     if _options.suffix:
44         for suffix in suffixes:
45             matches = Rankilist('{}-{}'.format(package_name, suffix))
46             lista.extend(matches)
47     if _options.prefix:
48         for prefix in prefixes:
49             matches = Rankilist('{}{}'.format(prefix, package_name))
50             lista.extend(matches)
51     if _options.suffix and _options.prefix:
52         for suffix in suffixes:
53             for prefix in prefixes:
```

```
54                 matches = Rankilist(
55                     '{}{}-{}'.format(prefix, package_name, suffix))
56                 lista.extend(matches)
57 
58     # ultimo = time.time()
59     lista = list(set(lista))
60     lista = sorted(lista, reverse=True)
61     for i in lista[:_options.amount]:
62         print i
```

Código A.4 – Protótipo do algoritmo de *Damerau-Levenshtein*.

```
1  # -*- coding: utf-8 -*-
2  """
3      Code to rank packages from a search in APT using Damerau Levenshtein
4  """
5  from apt import Cache
6  from pyxdameraulevenshtein import damerau_levenshtein_distance as ratio
7  from exact import Pack, _parser
8  from multiprocessing.pool import ThreadPool as Pool
9  
10 _MAX_PEERS = 20
11 
12 
13 def Thread_Rank(k):
14     """ Method to be done by a thread from the pool"""
15     pack = _args[0]
16     item = Pack()
17     item.name = k
18     item.ratio = ratio(pack, k)
19     return item
20 
21 
22 def Rankilist(pack):
23     """ Method to execute the algorithm """
24     cache = Cache()
25     if _options.single:
26         list_app = []
27         for k in cache:
28             item = Pack()
29             item.name = k.name
30             item.ratio = ratio(pack, k.name)
31             list_app.append(item)
32         return list_app
33     else:
34         _pool = Pool(processes=_MAX_PEERS)
```

```
35         result = _pool.map(Thread_Rank, cache._set)
36         return result
37 
38 if __name__ == '__main__':
39     (_options, _args) = _parser.parse_args()
40     package_name = _args[0]
41     suffixes = ['core', 'dev', 'commom', 'devel']
42     prefixes = ['lib']
43 
44     lista = Rankilist(package_name)
45     if _options.suffix:
46         for suffix in suffixes:
47             matches = Rankilist('{}-{}'.format(package_name, suffix))
48             lista.extend(matches)
49     if _options.prefix:
50         for prefix in prefixes:
51             matches = Rankilist('{}{}'.format(prefix, package_name))
52             lista.extend(matches)
53     if _options.suffix and _options.prefix:
54         for suffix in suffixes:
55             for prefix in prefixes:
56                 matches = Rankilist(
57                     '{}{}-{}'.format(prefix, package_name, suffix))
58                 lista.extend(matches)
59 
60     # ultimo = time.time()
61     lista = list(set(lista))
62     lista = sorted(lista)
63     for i in lista[:_options.amount]:
64         print i
```

Código A.5 – Protótipo do algoritmo de *Smith-Waterman*.

```
1  # -*- coding: utf-8 -*-
2  from apt import Cache
3  import swalign
4  from exact import Pack, _parser
5  from math import fabs
6  from multiprocessing.pool import ThreadPool as Pool
7  _MAX_PEERS = 20
8 
9 
10 def _cmp(x, y):
11     """ Ovewriting for compare method"""
12     if x.ratio[0] == y.ratio[0]:
13         if x.ratio[1] == y.ratio[1]:
```

```
14              if x.ratio[2] == y.ratio[2]:
15                  if x.ratio[3] == y.ratio[3]:
16                      return 0
17                  elif x.ratio[3] < y.ratio[3]:
18                      return -1
19                  else:
20                      return 1
21              elif x.ratio[2] < y.ratio[2]:
22                  return -1
23              else:
24                  return 1
25          elif x.ratio[1] < y.ratio[1]:
26              return 1
27          else:
28              return -1
29      elif x.ratio[0] < y.ratio[0]:
30          return -1
31      else:
32          return 1
33
34
35  def Thread_Align(k):
36      """ Method to be done by a thread from the pool"""
37      pack = _args[0]
38      item = Pack()
39      item.name = k
40      alinhamento = sw.align(pack, k)
41      item.ratio = (alinhamento.mismatches +
42                    fabs(len(pack) - len(k)) - alinhamento.matches) + len(pack)
43      return item
44
45
46  def SmithWaterman(pack):
47      """ Method to execute the algorithm """
48      cache = Cache()
49      if _options.single:
50          list_app = []
51          for k in cache:
52              item = Pack()
53              item.name = k.name
54              alinhamento = sw.align(pack, k.name)
55              item.ratio = [alinhamento.matches, fabs(len(pack) - len(k.name)),
56                            alinhamento.score, alinhamento.mismatches]
57              list_app.append(item)
58          return list_app
59      else:
60          _pool = Pool(processes=_MAX_PEERS)
```

```
61          result = _pool.map(Thread_Align, cache._set)
62          return result
63
64  if __name__ == '__main__':
65      (_options, _args) = _parser.parse_args()
66      package_name = _args[0]
67      suffixes = ['core', 'dev', 'commom', 'devel']
68      prefixes = ['lib']
69
70      match = 2
71      mismatch = -1
72      scoring = swalign.NucleotideScoringMatrix(match, mismatch)
73      sw = swalign.LocalAlignment(scoring)
74
75      _options.suffix = _options.prefix = False
76
77      lista = SmithWaterman(package_name)
78      if _options.suffix:
79          for suffix in suffixes:
80              matches = SmithWaterman('{}-{}'.format(package_name, suffix))
81              lista.extend(matches)
82      if _options.prefix:
83          for prefix in prefixes:
84              matches = SmithWaterman('{}{}'.format(prefix, package_name))
85              lista.extend(matches)
86      if _options.suffix and _options.prefix:
87          for suffix in suffixes:
88              for prefix in prefixes:
89                  matches = SmithWaterman(
90                      '{}{}-{}'.format(prefix, package_name, suffix))
91                  lista.extend(matches)
92
93      # ultimo = time.time()
94      lista = list(set(lista))
95      lista = sorted(lista, reverse=True, cmp=_cmp)
96      for i in lista[:_options.amount]:
97          print i
```

APÊNDICE B – Coleta de dados

B.1 Script de coleta de dados

Código B.1 – Script para automação da coleta de resultados

```
#!/usr/bin/python

import sys
import os
import csv
import subprocess
from multiprocessing.pool import ThreadPool
from pyexcel_ods import save_data

MAX = 150
_MAX_THREADS = 7
global b
data = {}
type_algorithm = {'kmp': [u'KMP', 1],
                  'rk': [u'Rabin-Karp', 0],
                  'ori': [u'Original', 2],
                  'levenshtein': [u'Levenshtein', 3],
                  'dice_coefficient': [u'Dice Coefficient', 4],
                  }

commands = sys.argv[1:]
if len(commands) == 0:
    commands = ['goo', 'gyt',
                'latexy', 'texlive',
                'libreoffice', 'libreoice',
                'firefox', 'firefoxy',
                'qtdeclarative', 'qwerty'
                ]
for cmds in commands:
    data[cmds] = [[u'Rabin-Karp'], [u'KMP'],
                  [u'Original'], [u'Levenshtein'], [u'Dice']]


def run(search):
    for i in xrange(0, MAX):
        cmd = './bin/apt search %s 2>> %s_%s.txt' % (search, search, b)
        subprocess.call(cmd, shell=True)
```

```
38
39
40 def process(file='firefox_test_apt-search-ori.txt'):
41     sheet = file.split('_')[0]
42     name = file.split('-')[-1].split('.')[0]
43     model = type_algorithm[name][1]
44     name = type_algorithm[name][0]
45     with open(file, 'rb') as csvfile:
46         spamreader = csv.reader(csvfile, delimiter=':')
47         print('Loading %s from %s') % (name, sheet)
48         for line in spamreader:
49             data[sheet][model].append(float(line[1]))
50     save_data('teste.ods', data)
51
52
53 if __name__ == '__main__':
54     branchs = ['test/apt-search-kmp',
55                'test/apt-search-rk',
56                'test/apt-search-ori',
57                'test/apt-search-levenshtein',
58                'test/apt-search-dice_coefficient',
59               ]
60     for branch in branchs:
61         subprocess.call('git checkout ' + branch, shell=True)
62         subprocess.call("make clean && make", shell=True)
63         b = branch.replace('/', '_')
64
65         _pool = ThreadPool(processes=_MAX_THREADS)
66         result = _pool.map(run, commands)
67         _pool.close()
68         _pool.join()
69
70     files = [f for f in os.listdir('.') if os.path.isfile(f)]
71     files = sorted(files)
72     for f in files:
73         try:
74             process(f)
75         except KeyError:
76             pass
```

B.2 Classe para coleta de tempo

Código B.2 – Classe para auxilio na coleta de tempo

```
1 #ifndef __TIMER__H__
2 #define __TIMER__H__
```

```
3
4  #include <chrono>
5
6  using namespace std::chrono;
7
8  class timer
9  {
10 private:
11     std::chrono::time_point<std::chrono::system_clock> start, stop;
12     std::chrono::duration<double> elapsed_seconds;
13 public:
14     timer(){}
15     ~timer(){}
16
17     void begin()
18     {
19         this->start = system_clock::now();
20     }
21
22     double end()
23     {
24         this->stop = std::chrono::system_clock::now();
25         this->elapsed_seconds = this->stop- this->start;
26         return elapsed_seconds.count();
27     }
28
29     double currenttime()
30     {
31         return elapsed_seconds.count();
32     }
33 };
34
35
36 #endif
```

APÊNDICE C – *Pull Requests*

C.1 Primeiro *Pull Request*

Código C.1 – *Diff* do primeiro *pull request*

```
commit 4d1cfa7a7670b877d9e33f8e23cb9d579a403570
Author: Luiz Oliveira <ziuloliveira@gmail.com>
Date:  Sun Aug 30 13:57:49 2015 -0300

    Included support to search packages into CLI ordered by:

     -  Alphabetic [default]
     -  Reverse Alphabetic
     -  Status
     -  Version

    Signed-off-by: Luiz Oliveira <ziuloliveira@gmail.com>

diff --git a/apt-private/private-cmndline.cc b/apt-private/private-cmndline.cc
index 1072b9a..1840386 100644
--- a/apt-private/private-cmndline.cc
+++ b/apt-private/private-cmndline.cc
@@ -57,6 +57,7 @@ static bool addArgumentsAPTCache(std::vector<CommandLine::Args>
     &Args, char cons
    {
       addArg('n', "names-only", "APT::Cache::NamesOnly", 0);
       addArg('f', "full", "APT::Cache::ShowFull", 0);
+      addArg(0, "order-by", "APT::Cache::OrderBy", CommandLine::HasArg);
    }
    else if (CmdMatches("show"))
    {
diff --git a/apt-private/private-output.cc b/apt-private/private-output.cc
index b8e6dec..cde8cb7 100644
--- a/apt-private/private-output.cc
+++ b/apt-private/private-output.cc
@@ -188,7 +188,7 @@ static std::string GetArchitecture(pkgCacheFile &CacheFile,
    pkgCache::PkgIterato
       return P.CurrentVer().Arch();
 }
                                    /*}}}*/
-static std::string GetShortDescription(pkgCacheFile &CacheFile, pkgRecords &
    records, pkgCache::PkgIterator P)/*{{{*/
```

```
35 +std::string GetShortDescription(pkgCacheFile &CacheFile, pkgRecords &records,
       pkgCache::PkgIterator P)/*{{{*/
36  {
37     pkgPolicy *policy = CacheFile.GetPolicy();
38
39 diff --git a/apt-private/private-output.h b/apt-private/private-output.h
40 index 4930fd9..0870143 100644
41 --- a/apt-private/private-output.h
42 +++ b/apt-private/private-output.h
43 @@ -25,6 +25,7 @@ APT_PUBLIC extern unsigned int ScreenWidth;
44
45  APT_PUBLIC bool InitOutput(std::basic_streambuf<char> * const out = std::cout.
        rdbuf());
46
47 +std::string GetShortDescription(pkgCacheFile &CacheFile, pkgRecords &records,
       pkgCache::PkgIterator P);
48  void ListSingleVersion(pkgCacheFile &CacheFile, pkgRecords &records,
49                         pkgCache::VerIterator const &V, std::ostream &out,
50                         std::string const &format);
51 diff --git a/apt-private/private-package-info.cc b/apt-private/private-package-
       info.cc
52 new file mode 100644
53 index 0000000..e7bb214
54 --- /dev/null
55 +++ b/apt-private/private-package-info.cc
56 @@ -0,0 +1,102 @@
57 +// Includes /**/
58 +#include <config.h>
59 +
60 +#include <apt-pkg/cachefile.h>
61 +#include <apt-pkg/pkgrecords.h>
62 +#include <apt-pkg/debversion.h>
63 +#include <apt-private/private-output.h>
64 +#include <apt-private/private-package-info.h>
65 +
66 +#include <algorithm>
67 +        /*}}}*/
68 +
69 +
70 +using namespace std;
71 +
72 +PackageInfo::PackageInfo(pkgCacheFile &CacheFile, pkgRecords &records,
73 +             pkgCache::VerIterator const &V, std::string formated_output)
74 +{
75 +   _formated_output = formated_output;
76 +
77 +   if(V)
```

```
78 +   {
79 +      pkgCache::PkgIterator const P = V.ParentPkg();
80 +      _name = P.Name();
81 +      _version = DeNull(V.VerStr());
82 +      _description = GetShortDescription(CacheFile, records, P);
83 +      _status = GetPackageStatus(CacheFile, V);
84 +   }
85 +
86 +}
87 +
88 +// PackageInfo::GetPackageStatus - Populate the package status /**/
89 +// ---------------------------------
90 +/* Returns the actual status of a package */
91 +PackageInfo::PackageStatus
92 +PackageInfo::GetPackageStatus(pkgCacheFile &CacheFile,
93 +             pkgCache::VerIterator const &V)
94 +{
95 +   pkgCache::PkgIterator const P = V.ParentPkg();
96 +   pkgDepCache * const DepCache = CacheFile.GetDepCache();
97 +   pkgDepCache::StateCache const &state = (*DepCache)[P];
98 +
99 +   PackageStatus Status = UNINSTALLED;
100 +   if (P->CurrentVer != 0)
101 +   {
102 +      if (P.CurrentVer() == V)
103 +      {
104 +    if (state.Upgradable() && state.CandidateVer != NULL)
105 +       Status = INSTALLED_UPGRADABLE;
106 +    else if (V.Downloadable() == false)
107 +       Status = INSTALLED_LOCAL;
108 +    else if(V.Automatic() == true && state.Garbage == true)
109 +       Status = INSTALLED_AUTO_REMOVABLE;
110 +    else if ((state.Flags & pkgCache::Flag::Auto) == pkgCache::Flag::Auto)
111 +       Status = INSTALLED_AUTOMATIC;
112 +    else
113 +       Status = INSTALLED;
114 +      }
115 +      else if (state.CandidateVer == V && state.Upgradable())
116 +      Status = UPGRADABLE;
117 +   }
118 +   else if (V.ParentPkg()->CurrentState == pkgCache::State::ConfigFiles)
119 +      Status = RESIDUAL_CONFIG;
120 +
121 +    return Status;
122 +}
123 +        /*}}}*/
124 +
```

```
125 +PackageInfo::SortBy
126 +PackageInfo::getOrderByOption ()
127 +{
128 +   std::string inString = _config->Find("APT::Cache::OrderBy","Alphabetic");
129 +   std::transform(inString.begin(), inString.end(), inString.begin(), ::tolower)
        ;
130 +   if (inString == "alphabetic") return PackageInfo::ALPHABETIC;
131 +   else if (inString == "reverse") return PackageInfo::REVERSEALPHABETIC;
132 +   else if (inString == "reverse alphabetic") return PackageInfo::
        REVERSEALPHABETIC;
133 +   else if (inString == "version") return PackageInfo::VERSION;
134 +   else if (inString == "status") return PackageInfo::STATUS;
135 +   return PackageInfo::ALPHABETIC;
136 +}
137 +
138 +bool OrderByStatus (const PackageInfo &a, const PackageInfo &b)
139 +{
140 +   if(a.status() == b.status())
141 +      return a.name() < b.name();
142 +   else
143 +      return a.status() < b.status();
144 +}
145 +
146 +bool OrderByAlphabetic (const PackageInfo &a, const PackageInfo &b)
147 +{
148 +   return a.name() < b.name();
149 +}
150 +
151 +bool OrderByVersion (const PackageInfo &a, const PackageInfo &b)
152 +{
153 +   int result = debVS.CmpVersion(a.version(),b.version());
154 +   if(result > 0)
155 +      return true;
156 +   else
157 +      return false;
158 +}
159 diff --git a/apt-private/private-package-info.h b/apt-private/private-package-
        info.h
160 new file mode 100644
161 index 0000000..5fb7f60
162 --- /dev/null
163 +++ b/apt-private/private-package-info.h
164 @@ -0,0 +1,55 @@
165 +#ifndef APT_PRIVATE_PACKAGE_INFO_H
166 +#define APT_PRIVATE_PACKAGE_INFO_H
167 +
168 +class PackageInfo
```

```
169 +{
170 +public:
171 +
172 +    typedef enum{
173 +        UNINSTALLED,
174 +        INSTALLED_UPGRADABLE,
175 +        INSTALLED_LOCAL,
176 +        INSTALLED_AUTO_REMOVABLE,
177 +        INSTALLED_AUTOMATIC,
178 +        INSTALLED,
179 +        UPGRADABLE,
180 +        RESIDUAL_CONFIG
181 +    } PackageStatus;
182 +
183 +    typedef enum{
184 +        ALPHABETIC,
185 +        REVERSEALPHABETIC,
186 +        STATUS,
187 +        VERSION
188 +    } SortBy;
189 +
190 +    PackageInfo(pkgCacheFile &CacheFile, pkgRecords &records,
191 +                    pkgCache::VerIterator const &V, std::string formated_output
    ="");
192 +
193 +    std::string format() const {return _format;}
194 +    std::string name() const {return _name;}
195 +    std::string version() const {return _version;}
196 +    PackageStatus status() const {return _status;}
197 +    std::string formated_output() const {return _formated_output;}
198 +
199 +    void set_formated_output(const std::string& formated_output){
    _formated_output = formated_output;}
200 +    void set_format(const std::string& format){_format = format;}
201 +
202 +    static SortBy getOrderByOption ();
203 +private:
204 +    std::string _name,
205 +            _formated_output,
206 +            _description,
207 +            _version,
208 +            _format = "${db::Status-Abbrev} ${Package} ${Version} ${Origin} ${
    Description}";
209 +    PackageStatus _status;
210 +
211 +    PackageStatus GetPackageStatus(pkgCacheFile &CacheFile, pkgCache::
    VerIterator const &V);
```

```
+};
+
+//Sort kinds
+bool OrderByStatus (const PackageInfo &a, const PackageInfo &b);
+bool OrderByVersion (const PackageInfo &a, const PackageInfo &b);
+bool OrderByAlphabetic (const PackageInfo &a, const PackageInfo &b);
+
+#endif
diff --git a/apt-private/private-search.cc b/apt-private/private-search.cc
index 6bce9ff..0d96893 100644
--- a/apt-private/private-search.cc
+++ b/apt-private/private-search.cc
@@ -2,29 +2,18 @@
 #include <config.h>

 #include <apt-pkg/cachefile.h>
-#include <apt-pkg/cacheset.h>
 #include <apt-pkg/cmndline.h>
 #include <apt-pkg/pkgrecords.h>
 #include <apt-pkg/policy.h>
 #include <apt-pkg/progress.h>
-#include <apt-pkg/cacheiterators.h>
-#include <apt-pkg/configuration.h>
-#include <apt-pkg/depcache.h>
-#include <apt-pkg/macros.h>
-#include <apt-pkg/pkgcache.h>

 #include <apt-private/private-cacheset.h>
-#include <apt-private/private-output.h>
 #include <apt-private/private-search.h>
+#include <apt-private/private-package-info.h>

-#include <string.h>
-#include <iostream>
 #include <sstream>
-#include <map>
-#include <string>
-#include <utility>
+#include <vector>

-#include <apti18n.h>
                                    /*}}}*/

 bool FullTextSearch(CommandLine &CmdL)            /*{{{*/
@@ -58,7 +47,7 @@ bool FullTextSearch(CommandLine &CmdL)            /*{{{*/

    bool const NamesOnly = _config->FindB("APT::Cache::NamesOnly", false);
```

```
259
260 -   std::map<std::string, std::string> output_map;
261 +   std::vector<PackageInfo> outputVector;
262
263     LocalitySortedVersionSet bag;
264     OpTextProgress progress(*_config);
265 @@ -111,18 +100,31 @@ bool FullTextSearch(CommandLine &CmdL)            /*{{{*/
266      PkgsDone[P->ID] = true;
267      std::stringstream outs;
268      ListSingleVersion(CacheFile, records, V, outs, format);
269 -    output_map.insert(std::make_pair<std::string, std::string>(
270 -        PkgName, outs.str()));
271 +    outputVector.emplace_back(CacheFile, records, V, outs.str());
272        }
273     }
274 +   switch(PackageInfo::getOrderByOption())
275 +   {
276 +     case PackageInfo::REVERSEALPHABETIC:
277 +    std::sort(outputVector.rbegin(), outputVector.rend(), OrderByAlphabetic);
278 +    break;
279 +     case PackageInfo::STATUS:
280 +    std::sort(outputVector.begin(), outputVector.end(), OrderByStatus);
281 +    break;
282 +     case PackageInfo::VERSION:
283 +    std::sort(outputVector.begin(), outputVector.end(), OrderByVersion);
284 +    break;
285 +     default:
286 +    std::sort(outputVector.begin(), outputVector.end(), OrderByAlphabetic);
287 +    break;
288 +   }
289     APT_FREE_PATTERNS();
290     progress.Done();
291
292 -   // FIXME: SORT! and make sorting flexible (alphabetic, by pkg status)
293 -   // output the sorted map
294 -   std::map<std::string, std::string>::const_iterator K;
295 -   for (K = output_map.begin(); K != output_map.end(); ++K)
296 -      std::cout << (*K).second << std::endl;
297 +   // output the sorted vector
298 +   for(auto k:outputVector)
299 +      std::cout << k.formated_output() << std::endl;
300 +
301
302     return true;
303  }
304 diff --git a/doc/apt.8.xml b/doc/apt.8.xml
305 index 18b97f5..7a6b487 100644
```

```
--- a/doc/apt.8.xml
+++ b/doc/apt.8.xml
@@ -53,7 +53,12 @@

      <varlistentry><term><option>search</option></term>
      <listitem><para><literal>search</literal> searches for the given
-     term(s) and display matching packages.
+     term(s) and display matching packages. Can be order by the following
    options:
+       <option>alphabetic</option>,
+       <option>reverse</option>,
+       <option>status</option> and
+       <option>version</option>.
+     Alphabetic order is the default.
      </para></listitem>
      </varlistentry>

diff --git a/test/integration/test-apt-cli-search b/test/integration/test-apt-cli
    -search
index e86661d..dc58175 100755
--- a/test/integration/test-apt-cli-search
+++ b/test/integration/test-apt-cli-search
@@ -15,11 +15,15 @@ fi

 DESCR='Some description that has a unusual word xxyyzz and aabbcc and a
     UPPERCASE'
 DESCR2='Some other description with the unusual aabbcc only'
+DESCR3='Some package description'
+DESCR4='an autogenerated dummy baz=0.1/installed'
 insertpackage 'unstable' 'foo' 'all' '1.0' '' '' "$DESCR
  Long description of stuff and such, with lines
  .
  and paragraphs and everything."
 insertpackage 'testing' 'bar' 'i386' '2.0' '' '' "$DESCR2"
+insertpackage 'stable' 'package' 'all' '1.5' '' '' "$DESCR3"
+insertinstalledpackage 'baz' 'all' '0.1'

 setupaptarchive

@@ -83,3 +87,61 @@ testsuccessequal "bar/testing 2.0 i386
 foo/unstable 1.0 all
   $DESCR
 " apt search -qq aabbcc
+
+# output is sorted in alphabetic reverse order
+testsuccessequal "package/stable 1.5 all
+  $DESCR3
```

```
350 +
351 +foo/unstable 1.0 all
352 +  $DESCR
353 +
354 +baz/now 0.1 all [installed,local]
355 +  $DESCR4
356 +
357 +bar/testing 2.0 i386
358 +  $DESCR2
359 +" apt search --order-by reverse -qq "\\w"
360 +
361 +# output is sorted by version
362 +testsuccessequal "bar/testing 2.0 i386
363 +  $DESCR2
364 +
365 +package/stable 1.5 all
366 +  $DESCR3
367 +
368 +foo/unstable 1.0 all
369 +  $DESCR
370 +
371 +baz/now 0.1 all [installed,local]
372 +  $DESCR4
373 +" apt search --order-by version -qq "\\w"
374 +
375 +# output is sorted by status
376 +testsuccessequal "bar/testing 2.0 i386
377 +  $DESCR2
378 +
379 +foo/unstable 1.0 all
380 +  $DESCR
381 +
382 +package/stable 1.5 all
383 +  $DESCR3
384 +
385 +baz/now 0.1 all [installed,local]
386 +  $DESCR4
387 +" apt search --order-by status -qq "\\w"
388 +
389 +# output is sorted by default
390 +testsuccessequal "bar/testing 2.0 i386
391 +  $DESCR2
392 +
393 +baz/now 0.1 all [installed,local]
394 +  $DESCR4
395 +" apt search --order-by some_strange_pattner -qq ba
396 +
```

```
+# output is sorted by Alphabetic (non case sense)
+testsuccessequal "bar/testing 2.0 i386
+  $DESCR2
+
+baz/now 0.1 all [installed,local]
+  $DESCR4
+" apt search --order-by Alphabetic -qq ba
\ No newline at end of file
```

Acima esta a diferença entre os arquivos originais do `APT` e as mudanças propostas na primeira contribuição.

C.2 Segundo *Pull Request*

Código C.2 – *Diff* do segundo *pull request*

```
diff --git a/.travis.yml b/.travis.yml
index 8be0285..9fad5cc 100644
--- a/.travis.yml
+++ b/.travis.yml
@@ -16,4 +16,4 @@ script:
  - make test
  - ./test/integration/run-tests -q
  - sudo adduser --force-badname --system --home /nonexistent --no-create-home --
      quiet _apt || true
- - sudo ./test/integration/run-tests -q
+ - sudo ./test/integration/run-tests -q
\ No newline at end of file
diff --git a/apt-private/private-cmndline.cc b/apt-private/private-cmndline.cc
index 7190fe5..5c81e6f 100644
--- a/apt-private/private-cmndline.cc
+++ b/apt-private/private-cmndline.cc
@@ -57,6 +57,8 @@ static bool addArgumentsAPTCache(std::vector<CommandLine::Args>
     &Args, char cons
    {
       addArg('n', "names-only", "APT::Cache::NamesOnly", 0);
       addArg('f', "full", "APT::Cache::ShowFull", 0);
+      addArg(0, "regex", "APT::Cache::UsingRegex", 0);
+      addArg(0, "order-by", "APT::Cache::OrderBy", CommandLine::HasArg);
    }
    else if (CmdMatches("show"))
    {
diff --git a/apt-private/private-output.cc b/apt-private/private-output.cc
index b8e6dec..cde8cb7 100644
--- a/apt-private/private-output.cc
+++ b/apt-private/private-output.cc
```

```
29 @@ -188,7 +188,7 @@ static std::string GetArchitecture(pkgCacheFile &CacheFile,
       pkgCache::PkgIterato
30          return P.CurrentVer().Arch();
31  }
32                                      /*}}}*/
33 -static std::string GetShortDescription(pkgCacheFile &CacheFile, pkgRecords &
       records, pkgCache::PkgIterator P)/*{{{*/
34 +std::string GetShortDescription(pkgCacheFile &CacheFile, pkgRecords &records,
       pkgCache::PkgIterator P)/*{{{*/
35  {
36     pkgPolicy *policy = CacheFile.GetPolicy();
37
38 diff --git a/apt-private/private-output.h b/apt-private/private-output.h
39 index 4930fd9..0870143 100644
40 --- a/apt-private/private-output.h
41 +++ b/apt-private/private-output.h
42 @@ -25,6 +25,7 @@ APT_PUBLIC extern unsigned int ScreenWidth;
43
44  APT_PUBLIC bool InitOutput(std::basic_streambuf<char> * const out = std::cout.
        rdbuf());
45
46 +std::string GetShortDescription(pkgCacheFile &CacheFile, pkgRecords &records,
       pkgCache::PkgIterator P);
47  void ListSingleVersion(pkgCacheFile &CacheFile, pkgRecords &records,
48                         pkgCache::VerIterator const &V, std::ostream &out,
49                         std::string const &format);
50 diff --git a/apt-private/private-package-info.cc b/apt-private/private-package-
       info.cc
51 new file mode 100644
52 index 0000000..2c8adac
53 --- /dev/null
54 +++ b/apt-private/private-package-info.cc
55 @@ -0,0 +1,102 @@
56 +// Includes /**/
57 +#include <config.h>
58 +
59 +#include <apt-pkg/cachefile.h>
60 +#include <apt-pkg/pkgrecords.h>
61 +#include <apt-pkg/debversion.h>
62 +#include <apt-private/private-output.h>
63 +#include <apt-private/private-package-info.h>
64 +
65 +#include <algorithm>
66 +        /*}}}*/
67 +
68 +
69 +using namespace std;
```

```
70 +
71 +PackageInfo::PackageInfo(pkgCacheFile &CacheFile, pkgRecords &records,
72 +             pkgCache::VerIterator const &V, std::string formated_output)
73 +{
74 +   _formated_output = formated_output;
75 +
76 +   if(V)
77 +   {
78 +      pkgCache::PkgIterator const P = V.ParentPkg();
79 +      _name = P.Name();
80 +      _version = DeNull(V.VerStr());
81 +      _description = GetShortDescription(CacheFile, records, P);
82 +      _status = GetPackageStatus(CacheFile, V);
83 +   }
84 +
85 +}
86 +
87 +// PackageInfo::GetPackageStatus - Populate the package status /**/
88 +// ------------------------------------
89 +/* Returns the actual status of a package */
90 +PackageInfo::PackageStatus
91 +PackageInfo::GetPackageStatus(pkgCacheFile &CacheFile,
92 +             pkgCache::VerIterator const &V)
93 +{
94 +   pkgCache::PkgIterator const P = V.ParentPkg();
95 +   pkgDepCache * const DepCache = CacheFile.GetDepCache();
96 +   pkgDepCache::StateCache const &state = (*DepCache)[P];
97 +
98 +   PackageStatus Status = UNINSTALLED;
99 +   if (P->CurrentVer != 0)
100 +   {
101 +      if (P.CurrentVer() == V)
102 +      {
103 +    if (state.Upgradable() && state.CandidateVer != NULL)
104 +       Status = INSTALLED_UPGRADABLE;
105 +    else if (V.Downloadable() == false)
106 +       Status = INSTALLED_LOCAL;
107 +    else if(V.Automatic() == true && state.Garbage == true)
108 +       Status = INSTALLED_AUTO_REMOVABLE;
109 +    else if ((state.Flags & pkgCache::Flag::Auto) == pkgCache::Flag::Auto)
110 +       Status = INSTALLED_AUTOMATIC;
111 +    else
112 +       Status = INSTALLED;
113 +      }
114 +      else if (state.CandidateVer == V && state.Upgradable())
115 +      Status = UPGRADABLE;
116 +   }
```

```
+   else if (V.ParentPkg()->CurrentState == pkgCache::State::ConfigFiles)
+      Status = RESIDUAL_CONFIG;
+
+    return Status;
+}
+        /*}}}*/
+
+PackageInfo::SortBy
+PackageInfo::getOrderByOption ()
+{
+   std::string inString = _config->Find("APT::Cache::OrderBy","Alphabetic");
+   std::transform(inString.begin(), inString.end(), inString.begin(), ::tolower)
    ;
+   if (inString == "alphabetic") return PackageInfo::ALPHABETIC;
+   else if (inString == "reverse") return PackageInfo::REVERSEALPHABETIC;
+   else if (inString == "reverse alphabetic") return PackageInfo::
    REVERSEALPHABETIC;
+   else if (inString == "version") return PackageInfo::VERSION;
+   else if (inString == "status") return PackageInfo::STATUS;
+   return PackageInfo::ALPHABETIC;
+}
+
+bool OrderByStatus (const PackageInfo &a, const PackageInfo &b)
+{
+   if(a.status() == b.status())
+      return a.name() < b.name();
+   else
+      return a.status() < b.status();
+}
+
+bool OrderByAlphabetic (const PackageInfo &a, const PackageInfo &b)
+{
+   return a.name() < b.name();
+}
+
+bool OrderByVersion (const PackageInfo &a, const PackageInfo &b)
+{
+   int result = debVS.CmpVersion(a.version(),b.version());
+   if(result > 0)
+      return true;
+   else
+      return false;
+}
diff --git a/apt-private/private-package-info.h b/apt-private/private-package-
    info.h
new file mode 100644
index 0000000..5fb7f60
```

```
--- /dev/null
+++ b/apt-private/private-package-info.h
@@ -0,0 +1,55 @@
+#ifndef APT_PRIVATE_PACKAGE_INFO_H
+#define APT_PRIVATE_PACKAGE_INFO_H
+
+class PackageInfo
+{
+public:
+
+    typedef enum{
+        UNINSTALLED,
+        INSTALLED_UPGRADABLE,
+        INSTALLED_LOCAL,
+        INSTALLED_AUTO_REMOVABLE,
+        INSTALLED_AUTOMATIC,
+        INSTALLED,
+        UPGRADABLE,
+        RESIDUAL_CONFIG
+    } PackageStatus;
+
+    typedef enum{
+        ALPHABETIC,
+        REVERSEALPHABETIC,
+        STATUS,
+        VERSION
+    } SortBy;
+
+    PackageInfo(pkgCacheFile &CacheFile, pkgRecords &records,
+                        pkgCache::VerIterator const &V, std::string formated_output
    ="");
+
+    std::string format() const {return _format;}
+    std::string name() const {return _name;}
+    std::string version() const {return _version;}
+    PackageStatus status() const {return _status;}
+    std::string formated_output() const {return _formated_output;}
+
+    void set_formated_output(const std::string& formated_output){
    _formated_output = formated_output;}
+    void set_format(const std::string& format){_format = format;}
+
+    static SortBy getOrderByOption ();
+private:
+    std::string _name,
+            _formated_output,
+            _description,
```

```
206 +           _version,
207 +           _format = "${db::Status-Abbrev} ${Package} ${Version} ${Origin} ${
    Description}";
208 +    PackageStatus _status;
209 +
210 +    PackageStatus GetPackageStatus(pkgCacheFile &CacheFile, pkgCache::
    VerIterator const &V);
211 +};
212 +
213 +//Sort kinds
214 +bool OrderByStatus (const PackageInfo &a, const PackageInfo &b);
215 +bool OrderByVersion (const PackageInfo &a, const PackageInfo &b);
216 +bool OrderByAlphabetic (const PackageInfo &a, const PackageInfo &b);
217 +
218 +#endif
219 diff --git a/apt-private/private-search.cc b/apt-private/private-search.cc
220 index 6bce9ff..47078de 100644
221 --- a/apt-private/private-search.cc
222 +++ b/apt-private/private-search.cc
223 @@ -2,31 +2,51 @@
224  #include <config.h>
225
226  #include <apt-pkg/cachefile.h>
227 -#include <apt-pkg/cacheset.h>
228  #include <apt-pkg/cmndline.h>
229  #include <apt-pkg/pkgrecords.h>
230  #include <apt-pkg/policy.h>
231  #include <apt-pkg/progress.h>
232 -#include <apt-pkg/cacheiterators.h>
233 -#include <apt-pkg/configuration.h>
234 -#include <apt-pkg/depcache.h>
235 -#include <apt-pkg/macros.h>
236 -#include <apt-pkg/pkgcache.h>
237
238  #include <apt-private/private-cacheset.h>
239 -#include <apt-private/private-output.h>
240  #include <apt-private/private-search.h>
241 +#include <apt-private/private-package-info.h>
242
243 -#include <string.h>
244 -#include <iostream>
245  #include <sstream>
246 -#include <map>
247 -#include <string>
248 -#include <utility>
249 +#include <vector>
250 +#include <algorithm>
```

```
251
252 -#include <apti18n.h>
253 +#define RABINKARPHASH(a, b, h, d) ((((h) - (a)*d) << 1) + (b))
254                                   /*}}}*/
255
256 +int RabinKarp(std::string StringInput, std::string Pattern) {
257 +   std::transform(StringInput.begin(), StringInput.end(), StringInput.begin(),
        ::tolower);
258 +   std::transform(Pattern.begin(), Pattern.end(), Pattern.begin(), ::tolower);
259 +   int string_length = StringInput.length();
260 +   int pattern_length = Pattern.length();
261 +
262 +   int mask, hash_input=0, hash_pattern=0;
263 +
264 +   /* Preprocessing */
265 +   /* computes mask = 2^(string_length-1) with
266 +      the left-shift operator */
267 +   mask = (1<<(string_length-1));
268 +
269 +   for (int i=0 ; i < string_length; ++i) {
270 +      hash_input = ((hash_input<<1) + StringInput.c_str()[i]);
271 +      hash_pattern = ((hash_pattern<<1) + Pattern.c_str()[i]);
272 +   }
273 +
274 +   /* Searching */
275 +   for (int i =0; i <= pattern_length-string_length; ++i) {
276 +      if (hash_input == hash_pattern && memcmp(StringInput.c_str(), Pattern.
        c_str() + i, string_length) == 0)
277 +   return i;
278 +      hash_pattern = RABINKARPHASH(Pattern.c_str()[i], Pattern.c_str()[i +
        string_length], hash_pattern, mask);
279 +   }
280 +
281 +   /* fould nothing*/
282 +   return -1;
283 +}
284 +
285  bool FullTextSearch(CommandLine &CmdL)            /*{{{*/
286  {
287     pkgCacheFile CacheFile;
288 @@ -45,20 +65,21 @@ bool FullTextSearch(CommandLine &CmdL)           /*{{{*/
289
290     // Compile the regex pattern
291     std::vector<regex_t> Patterns;
292 -   for (unsigned int I = 0; I != NumPatterns; ++I)
293 -   {
294 -      regex_t pattern;
```

```
295 -      if (regcomp(&pattern, CmdL.FileList[I + 1], REG_EXTENDED | REG_ICASE |
    REG_NOSUB) != 0)
296 +   if (_config->FindB("APT::Cache::UsingRegex",false))
297 +      for (unsigned int I = 0; I != NumPatterns; ++I)
298        {
299 -    APT_FREE_PATTERNS();
300 -    return _error->Error("Regex compilation error");
301 +    regex_t pattern;
302 +    if (regcomp(&pattern, CmdL.FileList[I + 1], REG_EXTENDED | REG_ICASE |
    REG_NOSUB) != 0)
303 +    {
304 +       APT_FREE_PATTERNS();
305 +       return _error->Error("Regex compilation error");
306 +    }
307 +    Patterns.push_back(pattern);
308        }
309 -      Patterns.push_back(pattern);
310 -   }
311
312     bool const NamesOnly = _config->FindB("APT::Cache::NamesOnly", false);
313
314 -   std::map<std::string, std::string> output_map;
315 +   std::vector<PackageInfo> outputVector;
316
317     LocalitySortedVersionSet bag;
318     OpTextProgress progress(*_config);
319 @@ -95,34 +116,64 @@ bool FullTextSearch(CommandLine &CmdL)          /*{{{*/
320        std::string const LongDesc = parser.LongDesc();
321
322        bool all_found = true;
323 -      for (std::vector<regex_t>::const_iterator pattern = Patterns.begin();
324 -       pattern != Patterns.end(); ++pattern)
325 +      if (_config->FindB("APT::Cache::UsingRegex",false))
326        {
327 -    if (regexec(&(*pattern), PkgName, 0, 0, 0) == 0)
328 -       continue;
329 -    else if (NamesOnly == false && regexec(&(*pattern), LongDesc.c_str(), 0, 0,
    0) == 0)
330 -       continue;
331 -    // search patterns are AND, so one failing fails all
332 -    all_found = false;
333 -    break;
334 +    for (std::vector<regex_t>::const_iterator pattern = Patterns.begin();
335 +         pattern != Patterns.end(); ++pattern)
336 +    {
337 +       if (regexec(&(*pattern), PkgName, 0, 0, 0) == 0)
338 +          continue;
```

```
339 +       else if (NamesOnly == false && regexec(&(*pattern), LongDesc.c_str(), 0,
        0, 0) == 0)
340 +          continue;
341 +       // search patterns are AND, so one failing fails all
342 +       all_found = false;
343 +       break;
344 +    }
345 +      }
346 +      else
347 +      {
348 +    for (unsigned int I = 0; I != NumPatterns; ++I)
349 +    {
350 +       if ((RabinKarp(CmdL.FileList[I + 1], P.Name()) >= 0) )
351 +       {
352 +          continue;
353 +       }
354 +       else if ( (NamesOnly == false) && (RabinKarp(CmdL.FileList[I + 1],
        LongDesc) >= 0))
355 +          continue;
356 +       all_found = false;
357 +       break;
358 +    }
359        }
360        if (all_found == true)
361        {
362     PkgsDone[P->ID] = true;
363     std::stringstream outs;
364     ListSingleVersion(CacheFile, records, V, outs, format);
365 -   output_map.insert(std::make_pair<std::string, std::string>(
366 -        PkgName, outs.str()));
367 +   outputVector.emplace_back(CacheFile, records, V, outs.str());
368        }
369     }
370 -   APT_FREE_PATTERNS();
371 +   switch(PackageInfo::getOrderByOption())
372 +   {
373 +      case PackageInfo::REVERSEALPHABETIC:
374 +    std::sort(outputVector.rbegin(), outputVector.rend(), OrderByAlphabetic);
375 +    break;
376 +      case PackageInfo::STATUS:
377 +    std::sort(outputVector.begin(), outputVector.end(), OrderByStatus);
378 +    break;
379 +      case PackageInfo::VERSION:
380 +    std::sort(outputVector.begin(), outputVector.end(), OrderByVersion);
381 +    break;
382 +      default:
383 +    std::sort(outputVector.begin(), outputVector.end(), OrderByAlphabetic);
```

```
384 +    break;
385 +   }
386 +   if (_config->FindB("APT::Cache::UsingRegex",false))
387 +      APT_FREE_PATTERNS();
388     progress.Done();
389
390 -   // FIXME: SORT! and make sorting flexible (alphabetic, by pkg status)
391 -   // output the sorted map
392 -   std::map<std::string, std::string>::const_iterator K;
393 -   for (K = output_map.begin(); K != output_map.end(); ++K)
394 -      std::cout << (*K).second << std::endl;
395 +   // output the sorted vector
396 +   for(auto k:outputVector)
397 +      std::cout << k.formated_output() << std::endl;
398
399     return true;
400  }
401 diff --git a/apt-private/private-search.h b/apt-private/private-search.h
402 index d478623..e50ff7c 100644
403 --- a/apt-private/private-search.h
404 +++ b/apt-private/private-search.h
405 @@ -6,6 +6,6 @@
406  class CommandLine;
407
408  APT_PUBLIC bool FullTextSearch(CommandLine &CmdL);
409 -
410 +int RabinKarp(std::string StringInput, std::string Pattern);
411
412  #endif
413 diff --git a/doc/apt.8.xml b/doc/apt.8.xml
414 index 18b97f5..7a6b487 100644
415 --- a/doc/apt.8.xml
416 +++ b/doc/apt.8.xml
417 @@ -53,7 +53,12 @@
418
419       <varlistentry><term><option>search</option></term>
420       <listitem><para><literal>search</literal> searches for the given
421 -     term(s) and display matching packages.
422 +     term(s) and display matching packages. Can be order by the following
      options:
423 +       <option>alphabetic</option>,
424 +       <option>reverse</option>,
425 +       <option>status</option> and
426 +       <option>version</option>.
427 +     Alphabetic order is the default.
428       </para></listitem>
429       </varlistentry>
```

```
430
431 diff --git a/test/integration/test-apt-cli-search b/test/integration/test-apt-cli
        -search
432 index e86661d..3343c1f 100755
433 --- a/test/integration/test-apt-cli-search
434 +++ b/test/integration/test-apt-cli-search
435 @@ -15,11 +15,15 @@ fi
436
437  DESCR='Some description that has a unusual word xxyyzz and aabbcc and a
        UPPERCASE'
438  DESCR2='Some other description with the unusual aabbcc only'
439 +DESCR3='Some package description'
440 +DESCR4='an autogenerated dummy baz=0.1/installed'
441  insertpackage 'unstable' 'foo' 'all' '1.0' '' '' "$DESCR
442   Long description of stuff and such, with lines
443   .
444   and paragraphs and everything."
445  insertpackage 'testing' 'bar' 'i386' '2.0' '' '' "$DESCR2"
446 +insertpackage 'stable' 'package' 'all' '1.5' '' '' "$DESCR3"
447 +insertinstalledpackage 'baz' 'all' '0.1'
448
449  setupaptarchive
450
451 @@ -55,7 +59,7 @@ testsuccessequal "foo/unstable 1.0 all
452  " apt search -qq aabbcc xxyyzz
453  testsuccessequal "foo/unstable 1.0 all
454    $DESCR
455 -" apt search -qq 'a+b+c+' 'i*xxy{0,2}zz'
456 +" apt search -qq --regex 'a+b+c+' 'i*xxy{0,2}zz'
457
458  # search is not case-sensitive by default
459  testsuccessequal "foo/unstable 1.0 all
460 @@ -63,7 +67,7 @@ testsuccessequal "foo/unstable 1.0 all
461  " apt search -qq uppercase
462  testsuccessequal "foo/unstable 1.0 all
463    $DESCR
464 -" apt search -qq 'up[pP]erc[Aa]se'
465 +" apt search -qq --regex 'up[pP]erc[Aa]se'
466
467  # search is done in the long description
468  testsuccessequal "foo/unstable 1.0 all
469 @@ -83,3 +87,61 @@ testsuccessequal "bar/testing 2.0 i386
470  foo/unstable 1.0 all
471    $DESCR
472  " apt search -qq aabbcc
473 +
474 +# output is sorted in alphabetic reverse order
```

```
475 +testequal "package/stable 1.5 all
476 +  $DESCR3
477 +
478 +foo/unstable 1.0 all
479 +  $DESCR
480 +
481 +baz/now 0.1 all [installed,local]
482 +  $DESCR4
483 +
484 +bar/testing 2.0 i386
485 +  $DESCR2
486 +" apt search --order-by reverse -qq "\\w" --regex
487 +
488 +# output is sorted by version
489 +testequal "bar/testing 2.0 i386
490 +  $DESCR2
491 +
492 +package/stable 1.5 all
493 +  $DESCR3
494 +
495 +foo/unstable 1.0 all
496 +  $DESCR
497 +
498 +baz/now 0.1 all [installed,local]
499 +  $DESCR4
500 +" apt search --order-by version -qq "\\w" --regex
501 +
502 +# output is sorted by status
503 +testequal "bar/testing 2.0 i386
504 +  $DESCR2
505 +
506 +foo/unstable 1.0 all
507 +  $DESCR
508 +
509 +package/stable 1.5 all
510 +  $DESCR3
511 +
512 +baz/now 0.1 all [installed,local]
513 +  $DESCR4
514 +" apt search --order-by status -qq "\\w" --regex
515 +
516 +# output is sorted by default
517 +testequal "bar/testing 2.0 i386
518 +  $DESCR2
519 +
520 +baz/now 0.1 all [installed,local]
521 +  $DESCR4
```

```
+" apt search --order-by some_strange_pattner -qq ba
+
+# output is sorted by Alphabetic (non case sense)
+testequal "bar/testing 2.0 i386
+  $DESCR2
+
+baz/now 0.1 all [installed,local]
+  $DESCR4
+" apt search --order-by Alphabetic -qq ba
```

C.3 Terceiro *Pull Request*

Código C.3 – *Diff do terceiro pull request*

```
commit aae1adc583ff0d7722968ea6202ca915931007e5
Author: Luiz Oliveira <ziuloliveira@gmail.com>
Date:   Wed Oct 28 02:30:08 2015 -0200

    Improved search when results are empty

    - If Results are empty, a new search will be done using Levenshtein
      to avoid typo spelling
    - If a regex is inserted to search, aonly a regex will be compilled

    Signed-off-by: Luiz Oliveira <ziuloliveira@gmail.com>

diff --git a/apt-private/private-package-info.cc b/apt-private/private-package-
    info.cc
index 2c8adac..96786e4 100644
--- a/apt-private/private-package-info.cc
+++ b/apt-private/private-package-info.cc
@@ -14,7 +14,7 @@
 using namespace std;

 PackageInfo::PackageInfo(pkgCacheFile &CacheFile, pkgRecords &records,
-             pkgCache::VerIterator const &V, std::string formated_output)
+          pkgCache::VerIterator const &V, std::string formated_output)
 {
    _formated_output = formated_output;

@@ -100,3 +100,8 @@ bool OrderByVersion (const PackageInfo &a, const PackageInfo
    &b)
    else
       return false;
 }
+
```

```
+bool OrderByDistance (const PackageInfo &a, const PackageInfo &b)
+{
+   return a.distance < b.distance;
+}
\ No newline at end of file
diff --git a/apt-private/private-package-info.h b/apt-private/private-package-info.h
index 5fb7f60..69ea08a 100644
--- a/apt-private/private-package-info.h
+++ b/apt-private/private-package-info.h
@@ -5,51 +5,55 @@ class PackageInfo
 {
 public:

-    typedef enum{
-        UNINSTALLED,
-        INSTALLED_UPGRADABLE,
-        INSTALLED_LOCAL,
-        INSTALLED_AUTO_REMOVABLE,
-        INSTALLED_AUTOMATIC,
-        INSTALLED,
-        UPGRADABLE,
-        RESIDUAL_CONFIG
-    } PackageStatus;
-
-    typedef enum{
-        ALPHABETIC,
-        REVERSEALPHABETIC,
-        STATUS,
-        VERSION
-    } SortBy;
-
-    PackageInfo(pkgCacheFile &CacheFile, pkgRecords &records,
-                        pkgCache::VerIterator const &V, std::string formated_output
="");
-
-    std::string format() const {return _format;}
-    std::string name() const {return _name;}
-    std::string version() const {return _version;}
-    PackageStatus status() const {return _status;}
-    std::string formated_output() const {return _formated_output;}
-
-    void set_formated_output(const std::string& formated_output){
_formated_output = formated_output;}
-    void set_format(const std::string& format){_format = format;}
-
-    static SortBy getOrderByOption ();
```

```
75 +   typedef enum{
76 +      UNINSTALLED,
77 +      INSTALLED_UPGRADABLE,
78 +      INSTALLED_LOCAL,
79 +      INSTALLED_AUTO_REMOVABLE,
80 +      INSTALLED_AUTOMATIC,
81 +      INSTALLED,
82 +      UPGRADABLE,
83 +      RESIDUAL_CONFIG
84 +   } PackageStatus;
85 +
86 +   typedef enum{
87 +      ALPHABETIC,
88 +      REVERSEALPHABETIC,
89 +      STATUS,
90 +       VERSION
91 +   } SortBy;
92 +   int distance;
93 +
94 +   PackageInfo(pkgCacheFile &CacheFile, pkgRecords &records,
95 +       pkgCache::VerIterator const &V, std::string formated_output="");
96 +
97 +   PackageInfo(){}
98 +   ~PackageInfo(){}
99 +   std::string format() const {return _format;}
100 +   std::string name() const {return _name;}
101 +   std::string version() const {return _version;}
102 +   PackageStatus status() const {return _status;}
103 +   std::string formated_output() const {return _formated_output;}
104 +
105 +   void set_formated_output(const std::string& formated_output){_formated_output
       = formated_output;}
106 +   void set_format(const std::string& format){_format = format;}
107 +
108 +   static SortBy getOrderByOption ();
109  private:
110 -    std::string _name,
111 -            _formated_output,
112 -            _description,
113 -            _version,
114 -            _format = "${db::Status-Abbrev} ${Package} ${Version} ${Origin} ${
       Description}";
115 -    PackageStatus _status;
116 -
117 -    PackageStatus GetPackageStatus(pkgCacheFile &CacheFile, pkgCache::
       VerIterator const &V);
118 +   std::string _name,
```

```
+      _formated_output,
+      _description,
+      _version,
+      _format = "${db::Status-Abbrev} ${Package} ${Version} ${Origin} ${
    Description}";
+   PackageStatus _status;
+
+   PackageStatus GetPackageStatus(pkgCacheFile &CacheFile, pkgCache::VerIterator
     const &V);
 };

 //Sort kinds
 bool OrderByStatus (const PackageInfo &a, const PackageInfo &b);
 bool OrderByVersion (const PackageInfo &a, const PackageInfo &b);
 bool OrderByAlphabetic (const PackageInfo &a, const PackageInfo &b);
+bool OrderByDistance (const PackageInfo &a, const PackageInfo &b);

 #endif
diff --git a/apt-private/private-search.cc b/apt-private/private-search.cc
index 5915667..a2a4418 100644
--- a/apt-private/private-search.cc
+++ b/apt-private/private-search.cc
@@ -16,6 +16,7 @@
 #include <vector>
 #include <algorithm>

+std::vector<PackageInfo> SimilarSearch(LocalitySortedVersionSet bag, std::vector
    <std::string> args);
 #define RABINKARPHASH(a, b, h, d) ((((h) - (a)*d) << 1) + (b))
                                   /*}}}*/

@@ -40,7 +41,7 @@ int RabinKarp(std::string StringInput, std::string Pattern) {
    /* Searching */
    for (int i =0; i <= pattern_length-string_length; ++i) {
       if (hash_input == hash_pattern && memcmp(StringInput.c_str(), Pattern.
           c_str() + i, string_length) == 0)
-   return i;
+    return i;
       hash_pattern = RABINKARPHASH(Pattern.c_str()[i], Pattern.c_str()[i +
           string_length], hash_pattern, mask);
    }

@@ -48,6 +49,52 @@ int RabinKarp(std::string StringInput, std::string Pattern) {
    return -1;
 }

+int levenshtein_distance(const std::string &s1, const std::string &s2)
```

```
+{
+   int s1len = s1.size();
+   int s2len = s2.size();
+
+   if (s2len > s1len)
+      return -1;
+   auto column_start = (decltype(s1len))1;
+
+   auto column = new decltype(s1len)[s1len + 1];
+   std::iota(column + column_start, column + s1len + 1, column_start);
+
+   for (auto x = column_start; x <= s2len; x++) {
+      column[0] = x;
+      auto last_diagonal = x - column_start;
+      for (auto y = column_start; y <= s1len; y++) {
+         auto old_diagonal = column[y];
+         auto possibilities = {
+            column[y] + 1,
+            column[y - 1] + 1,
+            last_diagonal + (s1[y - 1] == s2[x - 1]? 0 : 1)
+         };
+         column[y] = std::min(possibilities);
+         last_diagonal = old_diagonal;
+      }
+   }
+   auto result = column[s1len];
+   delete[] column;
+   return result;
+}
+
+bool identify_regex(std::vector<std::string> input)
+{
+   /*
+      not all characters can be included, as have
+      packages with the chars .-:
+   */
+   std::string reserver_regex = "^$*+?()[]{}\\|";
+
+   for(auto k:input)
+      if( reserver_regex.find(k) == std::string::npos )
+    return true;
+   return false;
+
+}
+
 bool FullTextSearch(CommandLine &CmdL)            /*{{{*/
 {
```

```
208     pkgCacheFile CacheFile;
209 @@ -66,7 +113,10 @@ bool FullTextSearch(CommandLine &CmdL)          /*{{{*/
210
211     // Compile the regex pattern
212     std::vector<regex_t> Patterns;
213 -   if (_config->FindB("APT::Cache::UsingRegex",false))
214 +   std::vector<std::string> args;
215 +   for (unsigned int I = 0; I != NumPatterns; ++I)
216 +      args.emplace_back( CmdL.FileList[I + 1]);
217 +   if ((_config->FindB("APT::Cache::UsingRegex",false)) || identify_regex(args))
218        for (unsigned int I = 0; I != NumPatterns; ++I)
219        {
220     regex_t pattern;
221 @@ -104,7 +154,7 @@ bool FullTextSearch(CommandLine &CmdL)          /*{{{*/
222     for ( ;V != bag.end(); ++V)
223     {
224        if (Done%500 == 0)
225 -         progress.Progress(Done);
226 +    progress.Progress(Done);
227        ++Done;
228
229        // we want to list each package only once
230 @@ -118,7 +168,7 @@ bool FullTextSearch(CommandLine &CmdL)          /*{{{*/
231        std::string const LongDesc = parser.LongDesc();
232
233        bool all_found = true;
234 -      if (_config->FindB("APT::Cache::UsingRegex",false))
235 +      if ((_config->FindB("APT::Cache::UsingRegex",false)) || identify_regex(
    args))
236        {
237     for (std::vector<regex_t>::const_iterator pattern = Patterns.begin();
238           pattern != Patterns.end(); ++pattern)
239 @@ -136,7 +186,7 @@ bool FullTextSearch(CommandLine &CmdL)          /*{{{*/
240        {
241     for (unsigned int I = 0; I != NumPatterns; ++I)
242     {
243 -      if ((RabinKarp(CmdL.FileList[I + 1], P.Name()) >= 0) )
244 +      if ((RabinKarp(CmdL.FileList[I + 1], P.Name()) >= 0) )
245        {
246           continue;
247        }
248 @@ -154,22 +204,31 @@ bool FullTextSearch(CommandLine &CmdL)          /*{{{*/
249     outputVector.emplace_back(CacheFile, records, V, outs.str());
250        }
251     }
252 -   switch(PackageInfo::getOrderByOption())
253 +   if(outputVector.size() == 0)
```

```
254     {
255 -     case PackageInfo::REVERSEALPHABETIC:
256 -    std::sort(outputVector.rbegin(), outputVector.rend(), OrderByAlphabetic);
257 -    break;
258 -     case PackageInfo::STATUS:
259 -    std::sort(outputVector.begin(), outputVector.end(), OrderByStatus);
260 -    break;
261 -     case PackageInfo::VERSION:
262 -    std::sort(outputVector.begin(), outputVector.end(), OrderByVersion);
263 -    break;
264 -     default:
265 -    std::sort(outputVector.begin(), outputVector.end(), OrderByAlphabetic);
266 -    break;
267 +     progress.Done();
268 +     outputVector = SimilarSearch(bag,args);
269 +     if(outputVector.size() > 10)
270 +    outputVector.resize(10);
271 +     std::cout << "None package fould. Maybe you are looking for one of those:"
       << std::endl;
272     }
273 -   if (_config->FindB("APT::Cache::UsingRegex",false))
274 +   else
275 +     switch(PackageInfo::getOrderByOption())
276 +     {
277 +    case PackageInfo::REVERSEALPHABETIC:
278 +       std::sort(outputVector.rbegin(), outputVector.rend(), OrderByAlphabetic);
279 +       break;
280 +    case PackageInfo::STATUS:
281 +       std::sort(outputVector.begin(), outputVector.end(), OrderByStatus);
282 +       break;
283 +    case PackageInfo::VERSION:
284 +       std::sort(outputVector.begin(), outputVector.end(), OrderByVersion);
285 +       break;
286 +    default:
287 +       std::sort(outputVector.begin(), outputVector.end(), OrderByAlphabetic);
288 +       break;
289 +     }
290 +   if ((_config->FindB("APT::Cache::UsingRegex",false)) || identify_regex(args))
291       APT_FREE_PATTERNS();
292     progress.Done();
293
294 @@ -380,3 +439,48 @@ bool DoSearch(CommandLine &CmdL)            /*{{{*/
295     return FullTextSearch(CmdL);
296  }
297
298 +std::vector<PackageInfo> SimilarSearch(LocalitySortedVersionSet bag, std::vector
       <std::string> args)
```

```
299 +{
300 +   pkgCacheFile CacheFile;
301 +   std::vector<PackageInfo> outputVector;
302 +   pkgCache *Cache = CacheFile.GetPkgCache();
303 +   pkgDepCache::Policy *Plcy = CacheFile.GetPolicy();
304 +   if (unlikely(Cache == NULL || Plcy == NULL))
305 +      return outputVector;
306 +   pkgRecords records(CacheFile);
307 +   std::vector<bool> PkgsDone(Cache->Head().PackageCount, false);
308 +   std::string format = "${color:highlight}${Package}${color:neutral}/${Origin}
      ${Version} ${Architecture}${ }${apt:Status}\n";
309 +   if (_config->FindB("APT::Cache::ShowFull",false) == false)
310 +      format += " ${Description}\n";
311 +   else
312 +      format += " ${LongDescription}\n";
313 +
314 +   for(auto V:bag)
315 +   {
316 +
317 +      // we want to list each package only once
318 +      pkgCache::PkgIterator const P = V.ParentPkg();
319 +      if (PkgsDone[P->ID] == true)
320 +    continue;
321 +
322 +      std::string PkgName = P.Name();
323 +      pkgCache::DescIterator Desc = V.TranslatedDescription();
324 +      pkgRecords::Parser &parser = records.Lookup(Desc.FileList());
325 +      std::string const LongDesc = parser.LongDesc();
326 +
327 +      for (auto arg:args)
328 +      {
329 +         int distance = levenshtein_distance(PkgName, arg);
330 +    if ((distance >=0) && (distance <= (int)PkgName.length()/2 ))
331 +    {
332 +       PkgsDone[P->ID] = true;
333 +       std::stringstream outs;
334 +       ListSingleVersion(CacheFile, records, V, outs, format);
335 +       outputVector.emplace_back(CacheFile, records, V, outs.str());
336 +       outputVector.back().distance = distance;
337 +    }
338 +      }
339 +   }
340 +   std::sort(outputVector.begin(), outputVector.end(), OrderByDistance);
341 +   return outputVector;
342 +}
343 diff --git a/apt-private/private-search.h b/apt-private/private-search.h
344 index 9c277e5..19a7586 100644
```

```
345 --- a/apt-private/private-search.h
346 +++ b/apt-private/private-search.h
347 @@ -10,5 +10,7 @@ APT_PUBLIC bool DoSearch(CommandLine &CmdL);
348  APT_PUBLIC void LocalitySort(pkgCache::VerFile ** const begin, unsigned long
        long const Count,size_t const Size);
349
350  APT_PUBLIC bool FullTextSearch(CommandLine &CmdL);
351 +bool identify_regex(std::vector<std::string> input);
352  int RabinKarp(std::string StringInput, std::string Pattern);
353 +int levenshtein_distance(const std::string &s1, const std::string &s2);
354  #endif
355 diff --git a/test/integration/test-apt-cli-search b/test/integration/test-apt-cli
       -search
356 index 3343c1f..5f7cf6a 100755
357 --- a/test/integration/test-apt-cli-search
358 +++ b/test/integration/test-apt-cli-search
359 @@ -43,7 +43,13 @@ foo/unstable 1.0 all
360  testsuccessequal "foo/unstable 1.0 all
361    $DESCR
362  " apt search -qq xxyyzz
363 -testempty apt search -qq --names-only xxyyzz
364 +
365 +#empty results
366 +testsuccessequal "None package fould. Maybe you are looking for one of those:"
       apt search -qq --names-only xxyyzz
367 +testsuccessequal "None package fould. Maybe you are looking for one of those:
368 +baz/now 0.1 all [installed,local]
369 +  $DESCR4
370 +" apt search -qq --names-only bez
371
372  # search name
373  testsuccessequal "foo/unstable 1.0 all
```

Anexos

ANEXO A – Figuras

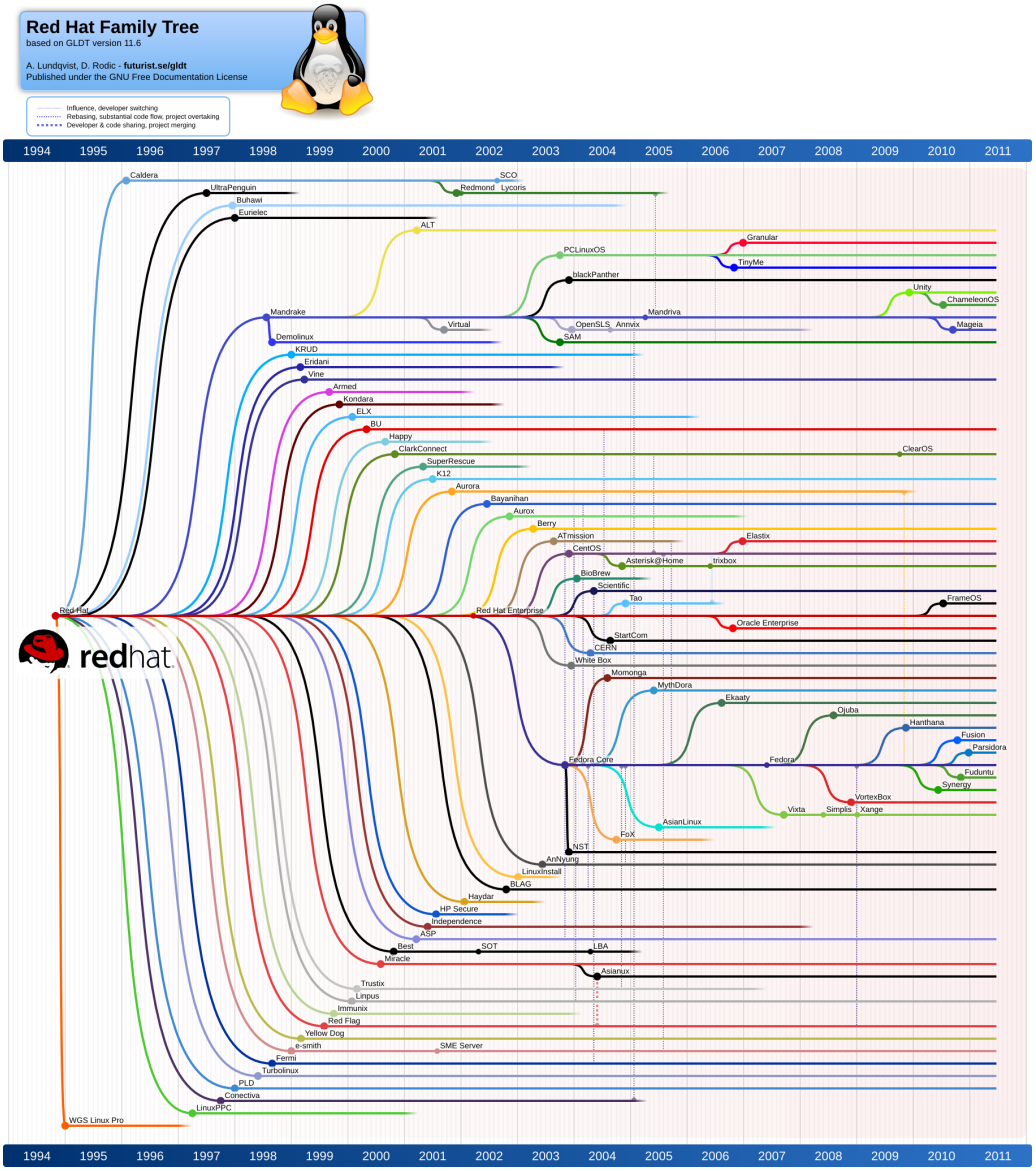

Figura 14 – Timeline ate 2011 da distribuição RedHat[1].

As figuras 14 e 15 apresentam as ramificações que surgiram cronologicamente nas distribuições Debian e RedHat. A linha na horizontal indica a *timeline* da distribuição. O desaparecimento da linha indica que a distribuição foi descontinuada. As linhas na vertical podem indicar uma ramificação (nova distribuição criada a partir de uma antiga) ou a fusão de distribuições. Linhas na horizontal contendo um circulo no meio do tracejado indicam que alguma distribuição alterou o seu nome (como foi o caso do S.u.S.E para SuSE no ano de 1998).

[1] **Fonte:** http://futurist.se/gldt/

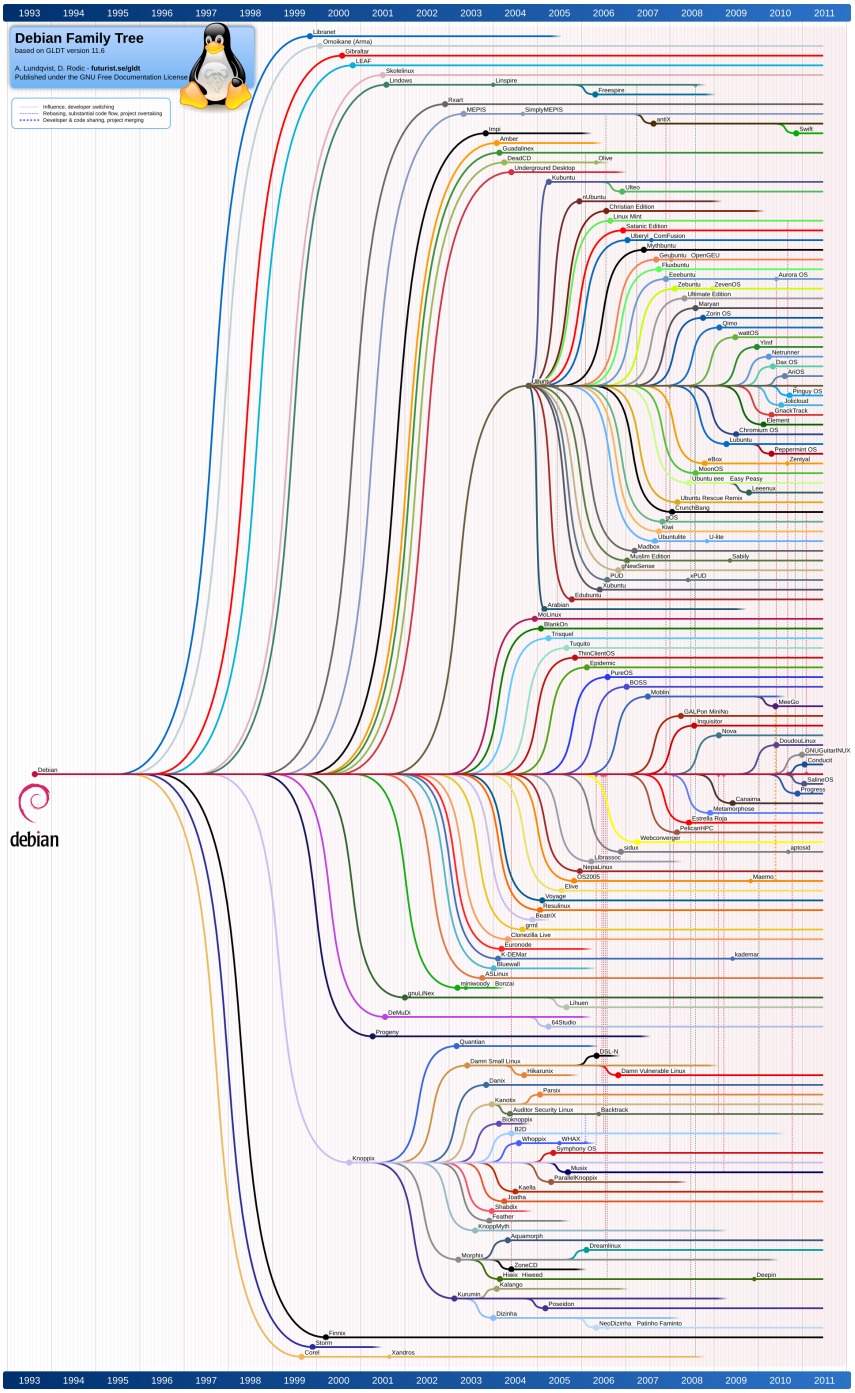

Figura 15 – Timeline ate 2011 da distribuição Debian[2].

Mesmo que de dificil visualização do nome das distribuições na imagem, o objetivo dela no trabalho é contextualizar a influencia que as distribuições *Debian* e *Red Hat* tem entre diversas distribuições e que novas funcionalidades nelas serão eventualmente adotadas nas suas herdeiras, aumentando a propagação de funcionalidades. Assim, uma nova funcionalidade no *Debian* certamente seria adotada na distribuição *Ubuntu* e suas ramificações, porém novas funcionalidades no *Ubuntu* demoram a serem copiadas para o *Debian*, quando são.

[2] **Fonte:** http://futurist.se/gldt/

As figuras são uma adaptação da imagem *GNU/Linux Distribution Timeline*, a imagem pode ser melhor vizualizada em <http://futurist.se/gldt/wp-content/uploads/11.06/gldt1106.png>.

ANEXO B – Guia de Estilo

Código B.1 – Guia de estilo do APT

```
Acronyms
~~~~~~~~
* dpkg is a 'word' the first d may be upper case - Dpkg
* APT is a proper Acronym, all upper case please.

Pkg - A Package
Ver - A version

Indenting, Comments, Etc
~~~~~~~~~~~~~~~~~~~~~~~
Would make Linus cry :P However it is what I prefer. 3 space indent,
8 space tab all braces on separate lines, function return on the same line
as the function, cases aligned with their code. The 'indent' options for
this style are:
   indent -bl -bli0 -di1 -i3 -nsc -ts8 -npcs -npsl

Each file gets a block at the top that should describe what the file does,
basically a summary of purpose along with any special notes and
attributions. The }}} and {{{ are folding marks if you have a folding
editor such as jed, the function separators are intended to give
a visual separate between functions for easier browsing of the larger files,
or indexed folding if you have such an editor.

Each file should have 1 or 0 primary include files, that include
file must always be the first include file included by the .cc. G++
#pragma interface/implementation is used, as well as anti-include-twice
#ifdefs.

Include files, since there are so many, get their own subdirectory off
the include search path, this is used consistently throughout all the code.
#include  should never be used for a global exported header file, only
local ones.

C++ Features
~~~~~~~~~~~~
Due to the legacy compiler heritage, exceptions, RTTI and name spaces are
not used. Templates are used *sparingly* since G++ has traditionally had
very weak support for them, this includes STL templates.

Namespaces will probably be put in the code sometime after G++ 3, which will
```

```
41 be a huge re-org again to make sanity, the majority of all nested things
42 will go away.
43
44 The C++ standard library's non parameterized types (string is included in
45 this) are used freely when appropriate.
46
47 The new C++ #include <iostream> (note the lack of a .h) is used for the
48 standard library, but not for my code.
49
50 Arguments and Ownership
51 ~~~~~~~~~~~~~~~~~~~~~~~
52 [much of the code follows this now]
53 These guidlines should be followed except in two cases.. the first
54 is where it makes no sense, such as in a casting operator and the second is to
55 retain API compatibility (this should be rare, since a change in the input
56 almost always designates a change in ownership rules).
57
58   * Pass by value or pass by reference should borrow the object from the
59     caller
60   * Pass by non-const reference may be used to indicate a OUT type variable
61   * Pass by pointer (except in the case where the pointer is really an array)
62     should be used when the object will be retained or ownership will be
63     transferred. Ownership transference should be rare and noted by a comment.
64   * Standard C things (FILE * etc) should be left as is.
65
66   * Return by references should indicate a borrowed object
67   * Return by pointer (except arrays) should indicate ownership is
68     transferred. Return by pointer should not be used unless ownership is
69     transferred.
70   * Return by pointer to variable indicates ownership transfer unless the
71     pointer is an 'input' parameter (designated generally by an =0,
72     indicating a default of 'none')
73
74 Non-ownership transferring arrays/lists should probably return an iterator
75 typedef or references..
```